МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«КУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет физики, математики, информатики
Кафедра алгебры, геометрии и теории обучения математике

КУРСОВАЯ РАБОТА
по дисциплине
Теория формальных языков и трансляций
на тему: РАЗРАБОТКА КОМПИЛЯТОРА МОДЕЛЬНОГО ЯЗЫКА ПРОГРАММИРОВАНИЯ
Вариант 16

Обучающегося 3 курса
очной формы обучения
направления подготовки
02.03.03 Математическое обеспечение
и администрирование
информационных систем
Направленность (профиль)
Проектирование информационных
систем и баз данных
Козявина Максима Сергеевича

Руководитель:
доцент кафедры АГиТОМ
Селиванова Ирина Васильевна

Допустить к защите:
_____ / _____
«_____» _____ 20_____г.

Курск, 2023

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В наши дни, несмотря на огромное количество разработанных языков программирования и соответствующих им компиляторов, процесс создания новых приложений в этой области не замедляется. Это обусловлено как прогрессом в технологии производства компьютерных систем, так и увеличивающимися требованиями к решению сложных задач. Существуют различные причины для разработки нового компилятора, такие как функциональные ограничения, отсутствие локализации или низкая эффективность существующих решений. Именно поэтому основы теории языков и формальных грамматик, а также практические методы разработки компиляторов играют особую роль в инженерном образовании в области информатики и вычислительной техники.

Создание собственного компилятора способствует не только пониманию основ языков программирования, но и дает возможность разработать специфичный язык программирования. Это поможет лучше понять основы информатики и алгоритмизации, а также применить формальную логику, теорию языков программирования и оптимизацию кода. Разработка компилятора также позволит изучить особенности работы с низкоуровневым кодом, углубить знания по работе с памятью и регистрами процессора.

Цель данной курсовой работы включает: получение навыков разработки программы компилятора на основе полученных теоретических знаний в области формальных языков и трансляций, позволяющего получить ассемблерный код и имеющего понятный пользовательский интерфейс, позволяющий визуализировать результаты работы всех этапов компиляции.

В рамках этой цели можно выделить следующие задачи:

– изучение особенностей разработки грамматик модельных языков программирования;

– разработка алгоритмов и методов программной реализации лексического, синтаксического, семантического анализов и перевода в польскую инверсную запись;

– визуализация этапов выполнения лексического, синтаксического и семантического анализов при практическом применении предложенного языка программирования;

– разработка алгоритмов перевода исходного кода в польскую инверсную запись и трансляции на ассемблерный язык;

– компиляция ассемблерного кода в исполняемый файл.

Объектом исследования выступают языки программирования.

Предметом исследования в курсовой работе выступают алгоритмы и методы разработки компиляторов языков программирования и их трансляции.

Работа выполнена на основе следующих структурных особенностей, указанных в варианте №16 задания к курсовой работе:

Выполняется задание согласно варианту 16:

Таблица 1 – вариант задания

№ Варианта	Задания											
	1	2	3	4	5	6	7	8	9	10	11	12
16	3	2	4	1	2	1	4	1	3	1	3	2

1. Структура программы

<программа> ::= {/ (<описание> | <оператор>) ; /}

2. Синтаксис команд описания данных

<описание> ::= dim <идентификатор> {, <идентификатор> } <тип>

3. Синтаксис идентификаторов

<идентификатор> ::= <буква> <буква> <непустая последовательность цифр>

4. Описание типов

(в порядке следования: целый, действительный, логический)

<тип> ::= % | ! | $

<оператор> ::= <составной> | <присваивания> | <условный> | <фиксированного_цикла> | <условного_цикла> | <ввода> | <вывода>

5. Синтаксис составного оператора

<составной> ::= «{» <оператор> { ; <оператор> } «}»

6. Синтаксис оператора присваивания

<присваивание> ::= <идентификатор> ass <выражение>

7. Синтаксис оператора условного перехода

<условный> ::= if (<выражение>) «{» <оператор> «}»{ elseif (<выражение>) «{» <оператор> «}»} { else «{» <оператор> «}»}

8. Синтаксис оператора фиксированного цикла

<фиксированного_цикла> ::= for <присваивания> to <выражение> «{» <оператор> “}”

9. Синтаксис оператора условного цикла

<условного_цикла> ::= do while <выражение> «{» <оператор> “}”

10. Синтаксис оператора ввода

<ввода> ::= read (<идентификатор> {, <идентификатор> })

11. Синтаксис оператора вывода

<вывода> ::= output (<выражение> { пробел <выражение> })

12. Признак начала комментария Признак конца комментария

/* */

Аннотация: в данной пояснительной записке содержится информация о разработке программы компилятора модельного языка программирования. Приложение включает в себя множество функций, таких как: лексический, синтаксический, семантический анализ и трансляция в польскую инверсную запись. Результатом работы приложения является поиск ошибок в написанном коде и перевод модельного языка в низкоуровневый язык с последующим преобразованием в исполняемый файл.

Ключевые слова: компилятор, язык программирования, лексический анализ, синтаксический анализ, семантический анализ, польская инверсная запись.

Annotation: this explanatory note contains information about the development of a model programming language compiler program. The application includes many functions, such as: lexical, syntactic, semantic analysis and translation into Polish inverse notation. The result of the application is to search for errors in the written code and translate the model language into a low-level language with subsequent conversion into an executable file.

Key words: compiler, programming language, lexical, syntactic, semantic analysis, reverse polish notation.

1 Разработка грамматики модельного языка программирования

1.1 Форма Бэкуса-Наура

<программа> ::= {/ (<описание> | <оператор>) ; /}

<описание> ::= dim <идентификатор> {, <идентификатор> } <тип>

<идентификатор> ::= <буква> <буква> <непустая последовательность цифр>

(в порядке следования: целый, действительный, логический)

<тип> ::= % | ! | $

<оператор> ::= <составной> | <присваивания> | <условный> | <фиксированного_цикла> | <условного_цикла> | <ввода> | <вывода>

<составной> ::= «{» <оператор> { ; <оператор> } «}»

<присваивание> ::= <идентификатор> ass <выражение>

<условный> ::= if (<выражение>) «{» <оператор> «}»{ elseif (<выражение>) «{» <оператор> «}»} { else «{» <оператор> «}»}

<фиксированного_цикла> ::= for <присваивания> to <выражение> «{» <оператор> “}”

<условного_цикла> ::= do while <выражение> «{» <оператор> «}»

<ввода> ::= read (<идентификатор> {, <идентификатор> })

<вывода> ::= output (<выражение> { пробел <выражение> })

<выражение> ::= <число> | <идентификатор> | not (<идентификатор>, <выражение>, <булево значение>) | - (<идентификатор>, <выражение>, <число>) | (<идентификатор>, <число>) <знак> (<идентификатор>, <число>)

<число> ::= <непустая последовательность цифр> | <непустая последовательность цифр>.<непустая последовательность цифр>

<булево значение> ::= true | false

<последовательность цифр> ::= {<цифра>}

<непустая последовательность цифр> ::= {/<цифра>/}

<знак> ::= + | - | * | / | > | < | <= | >= | = | and | or

<буква> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<начало_комментария> ::= </*>

<конец_комментария> ::= <*/>

1.2 Формальная грамматика

Грамматика представляет собой набор {T, N, P, S}, где T – множество терминальных символов:

```
T = { ;, +, -, *, /, >, >=, <, <=, =, !=, , «,», (, ), {, }, if, else, elseif, for, to, do, while, ass, dim, read, output, and, or, not, ID, TYPE, BOOL, NUM }
```

N – множество нетерминальных символов:

```
N = { PROG, CMD, IDS, I, EXPR, ASSIGN, FORCOND, COND1, COND, CALC, ANY }
```

S – начальный символ из набора нетерминалов:

```
S = PROG
```

P – правила формальной грамматики:

Программа:

```
PROG : CMD
```

Команды:

```
CMD : EXPR ; CMD | EXPR ; | ASSIGN ; CMD | ASSIGN ;
```

Список идентификаторов:

```
IDS : IDS , ID | I | I , ID
I : ID
```

Выражения:

```
EXPR : dim IDS TYPE | read ( IDS ) | output ( IDS ) | if ( COND ) { CMD } | if ( COND ) { CMD } else { CMD } | if ( COND ) { CMD } COND1 | do while ( COND ) { CMD } | for ( FORCOND ) { CMD }
```

Присвоение:

```
ASSIGN : ID ass CALC | ID ass ANY | ID ass I | ID ass COND
```

Условие фиксированного цикла:

```
FORCOND : ASSIGN to ANY | ASSIGN to I
```

Условия конструкций elseif, else:

```
COND1 : elseif ( COND ) { CMD } else { CMD } | elseif ( COND ) { CMD } COND1 | elseif ( COND ) { CMD }
```

Условие:

```
COND : ANY = ANY | ANY != ANY | ANY < ANY | ANY <= ANY | ANY > ANY | ANY >= ANY | ANY and ANY | ANY or ANY | not ANY | not ID | ANY = ID | ANY != ID | ANY < ID | ANY <= ID | ANY > ID | ANY >= ID | ANY and ID | ANY or ID | ID = ANY | ID != ANY | ID < ANY | ID <= ANY | ID > ANY | ID >= ANY | ID and ANY | ID or ANY | ID = ID | ID != ID | ID < ID | ID <= ID | ID > ID | ID >= ID | ID and ID | ID or ID
```

Вычисление значения:

```
CALC : ANY + ANY | ANY - ANY | ANY * ANY | ANY / ANY | ANY + ID | ANY - ID | ANY * ID | ANY / ID | ID + ANY | ID - ANY | ID * ANY | ID / ANY | ID + ID | ID - ID | ID * ID | ID / ID
```

Константа числового или булевого типа:

```
ANY : NUM | BOOL
```

2 Лексический, синтаксический и семантический анализы

2.1 Лексический анализ

2.1.1 Алгоритмы

Алгоритмы лексического анализа могут основываться на разделении на лексемы с помощью символов разделителей, чтением исходного кода посимвольно или с использованием регулярных выражений. Комментарии могут быть удалены либо же проигнорированы непосредственно при лексическом анализе.

В данном алгоритме используется посимвольное чтение исходного текста и последующая проверка символа или их комбинации на наличие соответствующей лексемы языка. При несоответствии выдаётся ошибка. Во время лексического анализа комментарии игнорируются.

2.1.2 Ошибки вывода

При лексическом анализе может быть выявлено 2 типа ошибок:

Первый тип – неверный идентификатор. Ошибка выдаётся если встречено слово, не являющееся ключевым словом, описанным в словаре `WORDS` класса `Lexer` и не соответствующее правилам наименования идентификаторов языка. В этом случае будет выведена ошибка `Lexer error: Unknown identifier "[имя идентификатора]" in line [номер строки].`

Второй тип – непредвиденный символ. Ошибка выдаётся, если встречен символ, не являющийся одним из символов-лексем языка, описанных в словаре `SYMBOLS` класса `Lexer`. В этом случае будет выведена ошибка `Lexer error: Unexpected symbol "[символ]" in line [номер строки].`

2.1.3 Хэш-таблица

Для разбиения исходного кода на лексемы используются встроенные в язык Python хэш-таблицы, именуемые словарями. В множествах и словарях языка Python используется метод открытой адресации. Он заключается в том, что в ячейки таблицы помещаются не указатели на списки, а сами пары ключ-значение. Значение зашифровывается хэш-функцией и при возникновении коллизии пара ключ-значение записываются в следующую пустую ячейку после той, которая получилась в результате работы хэш-функции. При извлечении данных из хэш-таблицы данные дополнительно сверяются с ключом, который так же хранится в хэш-таблице [3].

Хеш-функцией выступает метод `__hash__`, определённый в каждом хешируемом объекте языка Python. Эта функция может быть описана для любого типа данных, а для стандартных описана по стандарту. Для простых типов данных, например, int – результатом может быть само число, а для сложных может находиться комбинация хешей для составных частей и генератора случайных чисел, который гарантирует одинаковые значения для одного типа данных в рамках одного запуска программы.

2.1.4 Реализация лексического анализа

Лексический анализ реализует модуль `lexer` и, в частности, класс `Lexer`, содержащий хэш-таблицы символов и ключевых слов – `SYMBOLS` и `WORDS`. Входными данными для класса является текстовый поток input_stream из файла текста программы. Метод `getc()` получает следующий символ из потока input_stream. Метод `set_error()` выставляет сообщение об ошибке. Метод `next_token()` считывает следующую лексему посимвольно, определяет тип лексемы, игнорирует комментарии и вызывает метод `set_error()` при возникновении ошибки. В результате в поле `symbol`

выводится тип лексемы и в поле `value` значение для целочисленных, дробных и булевых литералов или имя идентификатора.

2.1.5 Тестирование

Задание. Найти значение функции `y = kx+b`.

Тест 1. Программа написана без ошибок

Исходный код программы на модульном языке программирования:

```
dim xx1, yy1, kk1, bb1, rr1 !;
rr1 ass xx1 * kk1;
rr1 ass rr1 + bb1;
yy1 ass rr1;
```

Результат работы лексического анализа представлен на рисунке 1.

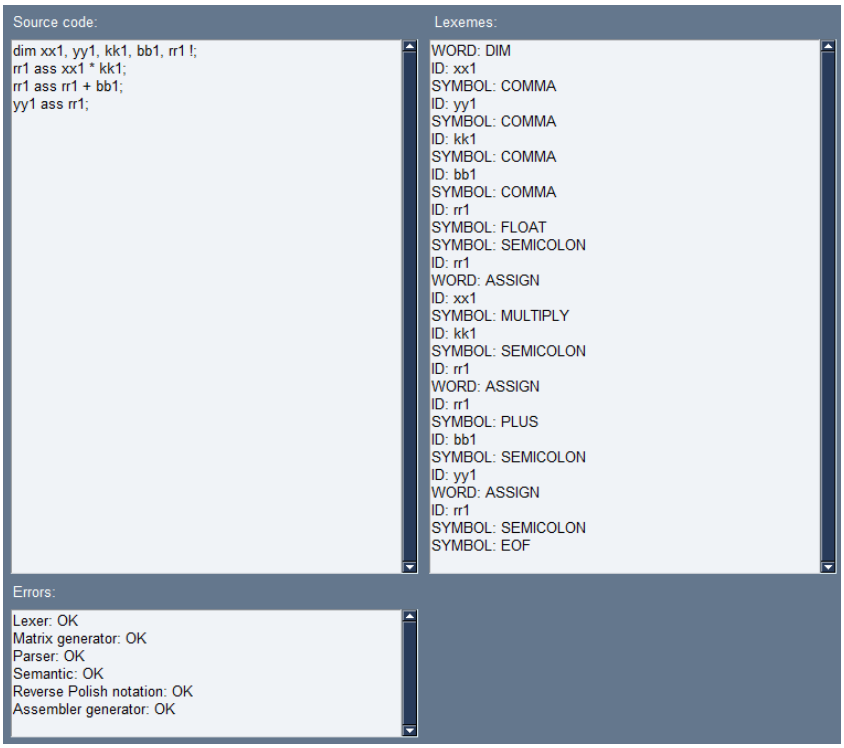

Рисунок 1 – Тест 1 алгоритма лексического анализа

Тест 1. Программа написана с лексической ошибкой (не существует оператора или идентификатора `as)`.

Исходный код программы на модульном языке программирования:

```
dim xx1, yy1, kk1, bb1, rr1 !;
rr1 as xx1 * kk1;
rr1 ass rr1 + bb1;
yy1 ass rr1;
```

Результат работы лексического анализа представлен на рисунке 2.

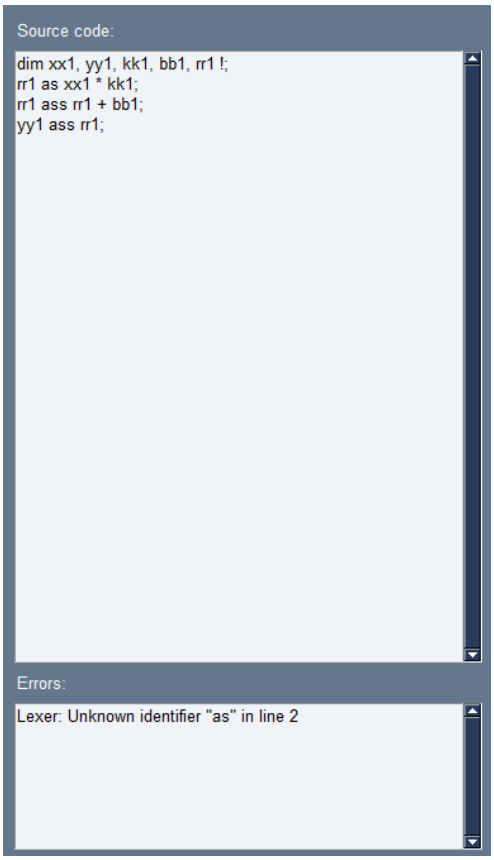

Рисунок 2 – Тест 2 алгоритма лексического анализа

2.2 Синтаксический анализ

2.2.1 Алгоритмы

Существует несколько алгоритмов синтаксического анализа:

Алгоритм рекурсивного спуска – алгоритм нисходящего синтаксического анализа, реализуемый путём взаимного вызова процедур парсинга, где каждая процедура соответствует одному из правил контекстно-свободной грамматики или БНФ. Применения правил последовательно, слева-

направо поглощают токены, полученные от лексического анализатора. Это один из самых простых алгоритмов парсинга, который является также малоэффективным т.к. может обрабатывать большое количество текста недостаточно быстро и может обрабатывать не все грамматики.

Алгоритм сдвиг-свёртки используется в данном алгоритме. Он используется для грамматики операторного предшествования. Для моделирования его работы необходима входная цепочка символов и стек символов, в котором автомат может обращаться не только к самому верхнему символу, но и к некоторой цепочке символов на вершине стека. Также необходимо построить матрицу операторного предшествования [1].

Этот алгоритм для заданной КС-грамматики можно описать следующим образом:

1. Поместить в верхушку стека символ «начало строки», считывающую головку МП-автомата поместить в начало входной цепочки. В конец входной цепочки надо дописать символ «конец строки».

2. В стеке ищется самый верхний терминальный символ sj при этом сам символ sj остается в стеке. Из входной цепочки берется текущий символ ai (справа от считывающей головки МП-автомата).

3. Если символ sj – это символ начала строки, а символ ai – символ конца строки, то алгоритм завершен, входная цепочка символов разобрана.

4. В матрице предшествования ищется клетка на пересечении строки, помеченной символом sj, и столбца, помеченного символом ai.

5. Если клетка, пустая, то значит, входная строка символов не принимается, алгоритм прерывается и выдает сообщение об ошибке.

6. Если клетка, содержит символ “=.” или “<.” то необходимо выполнить перенос. При выполнении переноса текущий входной символ ai помещается на верхушку стека, считывающая головка сдвигается на одну позицию вправо. После этого надо вернуться к шагу 2.

7. Если клетка, содержит символ “.>”, то необходимо произвести свертку. Для выполнения свертки из стека выбираются все терминальные символы, связанные отношением “=.”, начиная от вершины стека, а также все нетерминальные символы, лежащие в стеке рядом с ними. Эти символы вынимаются из стека и собираются в цепочку.

8. Во всем множестве правил грамматики ищется правило, у которого правая часть совпадает с цепочкой. Если правило найдено, то в стек помещается нетерминальный символ из левой части правила, иначе, если правило не найдено, это значит, что входная строка символов не принимается, алгоритм прерывается и выдает сообщение об ошибке. После выполнения свертки необходимо вернуться к шагу 2.

В данном алгоритме матрица операторного предшествования строится автоматически из правил грамматики перед выполнением алгоритма сдвиг-свёртки.

2.2.2 Ошибки вывода

При построении матрицы операторного предшествования может быть выведены ошибки:

1) при встрече неизвестного символа – `unknown symbol "[символ]"`;
2) при обнаружении неверно построенного файла правил вывода – `wrong file formatting`;
3) при конфликте в правилах вывода с указанием ошибки.

При выполнении алгоритма сдвиг-свёртки:

1) при отсутствии правила в исходных правилах вывода – `Unable to locate rule [правило]`;

2) при отсутствии связи в таблице операторного предшествования – `unknown construction [символ 1] [символ 2] in [фрагмент кода]`.

2.2.3 Реализация синтаксического анализа

Для построения матрицы операторного предшествования в класс `Matrix` передаётся текстовый поток `input_stream` из текстового файла, в котором описаны все терминальные символы, все имена правил и перечислены все правила вывода, причём, имя и само правило разделяются символом «:». Все терминальные символы должны быть разделены пробелом. Метод `print_matrix()` позволяет вывести результат на экран, а метод `generate()` генерирует матрицу операторного предшествования. Строятся множества крайних левых и крайних правых символов для всех символов и только для терминальных символов. После по этим множествам составляется матрица операторного предшествования.

На рисунке 3 представлена полученная матрица операторного предшествования для данного модельного языка программирования.

	;	+	-	*	/	>	>=	<	<=	=	!=	,	(	)	{	}	if	else	elseiffor	to	do	while	ass	dim	read	outputand	or	not	ID	TYPE	BOOL	NUM	/e/
;	<.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.>	<.	.	.	<.	.	<.	.	.	<.	<.	<.	.	.	.	<.	.	.	.	.>
+	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
-	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
*	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
/	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
>	.>	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
>=	.>	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
<	.>	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
<=	.>	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
=	.>	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
!=	.>	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
,	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.
(	.	.	.	.	.	<.	<.	<.	<.	<.	<.	<.	.	=.	.	.	.	.	.	<.	.	.	.	.	.	.	<.	<.	<.	<.	.	<.	.	<.	.
)	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
{	<.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=.	<.	.	.	<.	.	<.	.	.	<.	<.	<.	.	.	.	<.	.	.	.	.
}	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=.	<.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
if	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
else	.	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
elseif	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
for	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
to	.	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	<.	<.	.
do	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.
while	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
ass	.>	<.	<.	<.	<.	<.	<.	<.	<.	<.	<.	<.	.	.	.	.	.	.	.	.	.>	.	.	.	.	<.	<.	<.	<.	.	<.	.	<.	<.	.
dim	.	.	.	.	.	.	.	.	.	.	.	<.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	<.	=.	.	.	.
read	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
output	.	.	.	.	.	.	.	.	.	.	.	.	=.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
and	.>	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
or	.>	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
not	.>	.	.	.	.	.	.	.	.	.	.	.	.	.>	.	.	.	.	.	.	.>	.	.	.	.	.	.	.	.	.	=.	.	<.	<.	.
ID	.>	=.	=.	=.	=.	=.	=.	=.	=.	=.	=.	.>	.	.>	.	.	.	.	.	.	.>	.	.	=.	.	.	.	=.	=.	.	.	.>	.	.	.
TYPE	.>	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
BOOL	.>	.>	.>	.>	.>	.>	.>	.>	.>	.>	.>	.	.	.>	.	.	.	.	.	.	.>	.	.	.	.	.	.	.>	.>	.	.	.	.	.	.
NUM	.>	.>	.>	.>	.>	.>	.>	.>	.>	.>	.>	.	.	.>	.	.	.	.	.	.	.>	.	.	.	.	.	.	.>	.>	.	.	.	.	.	.
/b/	<.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	<.	.	.	.	<.	.	<.	.	.	<.	<.	<.	.	.	.	<.	.	.	.	.

Рисунок 3 – Матрица операторного предшествования

Эта матрица передаётся для выполнения алгоритма сдвиг-свёртки в класс `Parser`. Перед выполнением сдвиг-свёртки все нетерминалы в правилах вывода заменяются на один нетерминал «E». При вызове метода `check` выполняется проверка лексем на корректность по матрице операторного предшествования. Метод `get_t` возвращает крайний терминал

стека, а при передаче аргумента `get_t(n)` может вернуть *n*-ый символ с конца стека. Для вывода ошибок используется метод `set_error`.

2.2.4 Тестирование

Задание. Проверить, что треугольник со сторонами *a, b, c* существует.

Тест 1. Программа написана без ошибок.

Исходный код программы на модульном языке программирования:

```
dim aa1, bb1, cc1, ss1 !;
dim rr1 $;
rr1 ass true;
read(aa1, bb1, cc1);
ss1 ass aa1 + bb1;
if (ss1 < cc1) {
    rr1 ass false;
};
ss1 ass aa1 + cc1;
if (ss1 < bb1) {
    rr1 ass false;
};
ss1 ass bb1 + cc1;
if (ss1 < aa1) {
    rr1 ass false;
};
output(rr1);
```

Результат работы синтаксического анализа представлен на рисунке 4.

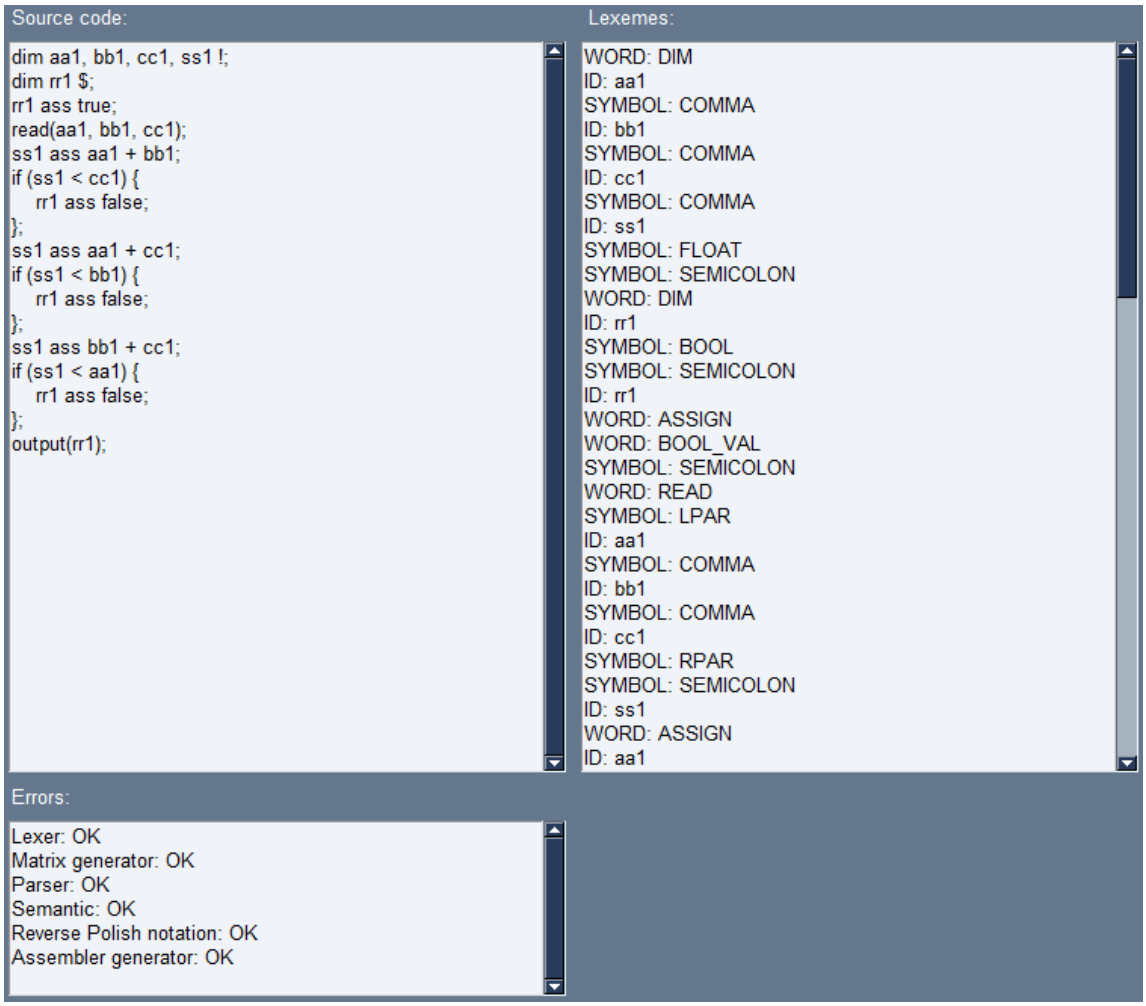

Рисунок 4 – Тест 1 алгоритма синтаксического анализа

Тест 2. Программа написана с синтаксической ошибкой (`elseif` не может идти после «`;`»). Этот фрагмент указан ниже.

```
dim aa1, bb1, cc1, ss1 !;
dim rr1 $;
rr1 ass true;
ss1 ass aa1 + bb1;
elseif (ss1 < cc1) {
    rr1 ass false;
};
...
```

Результат работы синтаксического анализа представлен на рисунке 5.

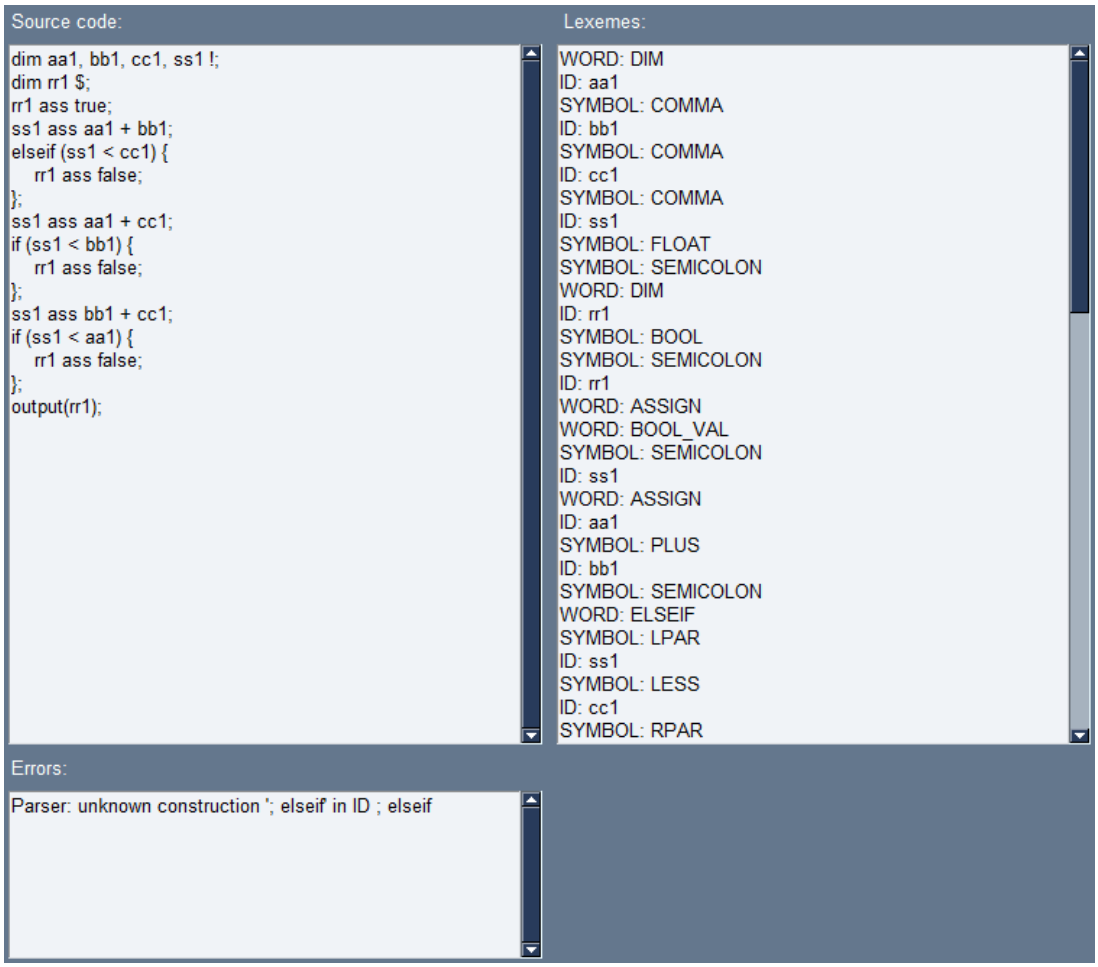

Рисунок 5 – Тест 2 алгоритма синтаксического анализа

Тест 3. Программа написана с синтаксической ошибкой (не существует правила грамматики для фрагмента кода `ss1 < cc1 = 1`). Этот фрагмент указан ниже.

```
dim aa1, bb1, cc1, ss1 !;
dim rr1 $;
rr1 ass true;
ss1 ass aa1 + bb1;
if (ss1 < cc1 = 1) {
    rr1 ass false;
};
...
```

Результат работы синтаксического анализа представлен на рисунке 6.

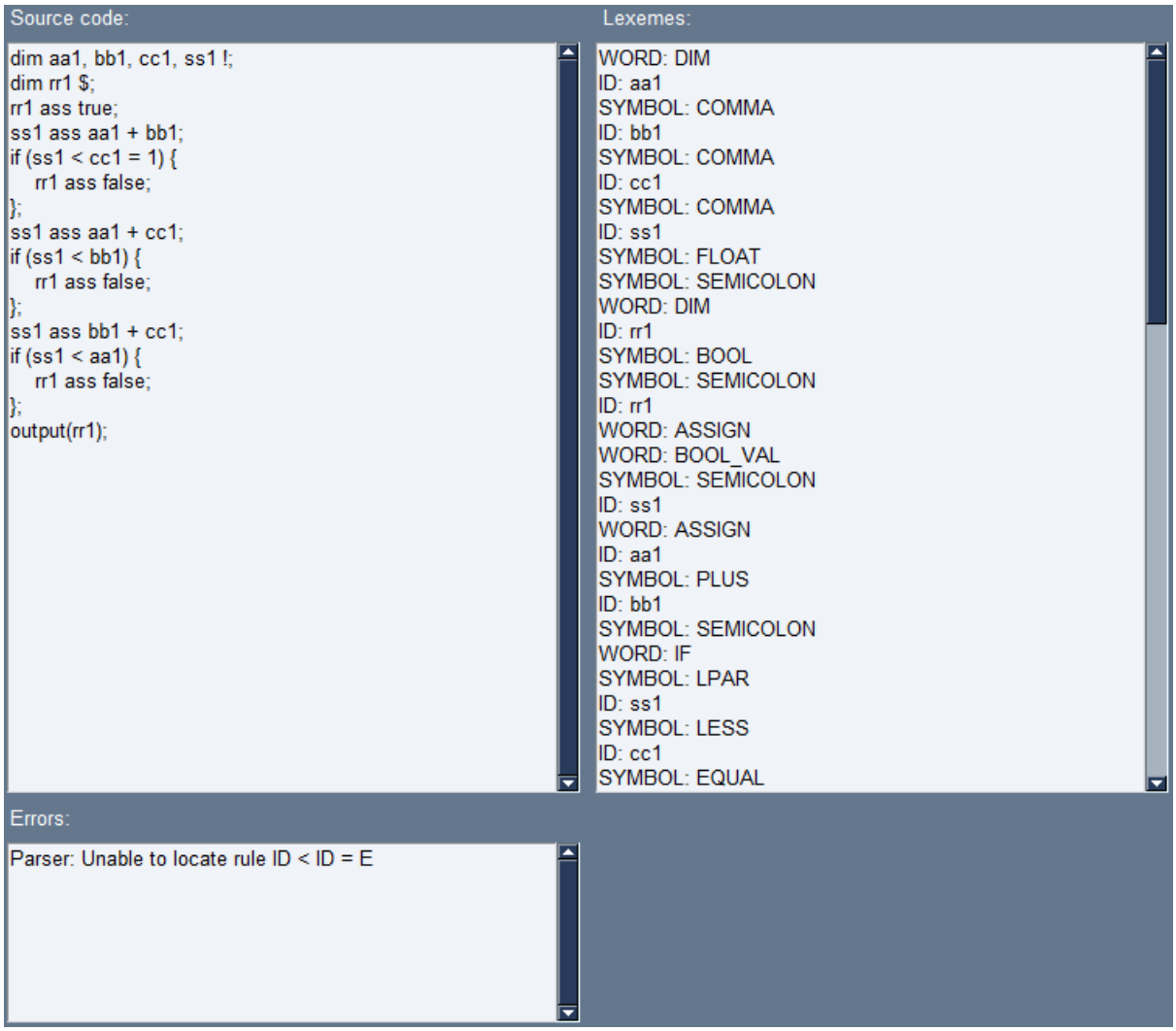

Рисунок 6 – Тест 3 алгоритма синтаксического анализа

2.3 Семантический анализ

Семантические правила, проверяемые при анализе:

1. Переменная, использованная в ходе выполнения алгоритма на модельном языке программирования, должна быть определена конструкцией `dim`.

2. При присвоении переменной литерала, значения переменной или результата операции необходимо совпадение типов левой и правой части присвоения.

2.3.1 Алгоритм

Для семантического анализа используется простой алгоритм поиска в лексемах определённых конструкций модельного языка программирования с последующей проверкой правильности типов и определения переменных в этих конструкциях.

2.3.2 Ошибки вывода

При попытке обращения к переменной, которая не была определена конструкцией `dim`, будет выведена ошибка `Undeclared variable [имя переменной]`.

При попытке присвоить переменной литерал несоответствующего типа или значение переменной несоответствующего типа будет выведена ошибка `Wrong type: [имя переменной] is [тип переменной] unable to assign [тип, который пытался присвоить пользователь]`.

2.3.3 Реализация семантического анализа

Метод check проверяет входную цепочку лексем на соответствие типов и ведёт учёт переменных, определённых в исходном коде. Считывание происходит поэлементно. Алгоритм ищет цепочку вида `dim [имена переменных] [тип]` для занесения переменной в список определённых переменных, цепочки вида `[переменная] ass [переменная]`, `[переменная] ass [литерал]` или `[переменная] ass [переменная или литерал] [операция] [переменная или литерал]` для контроля соответствия типов. Любые другие вхождения переменных будут сопоставляться со списком определённых в программе переменных. В классе существуют поля-массивы операций `RETURN_BOOL` и `RETURN_NUM`, которые определяют какой тип возвращают соответствующие операции. Для вывода ошибок используется метод `set_error`.

2.3.4 Тестирование

Задание. Рассчитать сумму чётных чисел от 2 до *n*.

Тест 1. Программа написана без ошибок.

Исходный код программы на модульном языке программирования:

```
dim ii1, ss1, tm1, nn1 !;
read(nn1);
nn1 ass nn1/2;
nn1 ass nn1 + 1;
for (ii1 ass 1 to nn1) {
    tm1 ass ii1 * 2;
    ss1 ass ss1 + tm1;
};
```

Результат работы алгоритма семантического анализа представлен на рисунке 7.

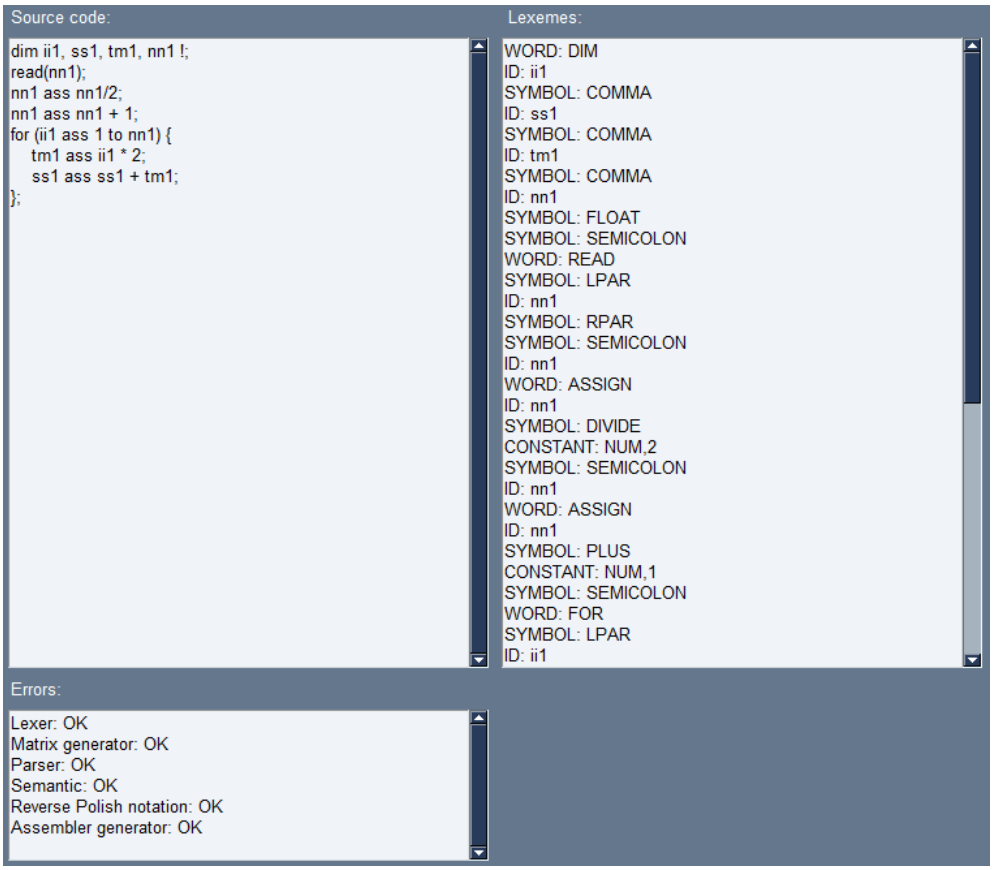

Рисунок 7 – Тест 1 алгоритма семантического анализа

Тест 2. Программа написана с семантической ошибкой (невозможно присвоить переменной типа FLOAT константу типа BOOL).

Исходный код программы на модульном языке программирования:

```
dim ii1, ss1, tm1, nn1 !;
read(nn1);
nn1 ass nn1/2;
nn1 ass nn1 + 1;
for (ii1 ass 1 to nn1) {
    tm1 ass true;
    ss1 ass ss1 + tm1;
};
```

Результат работы алгоритма семантического анализа представлен на рисунке 8.

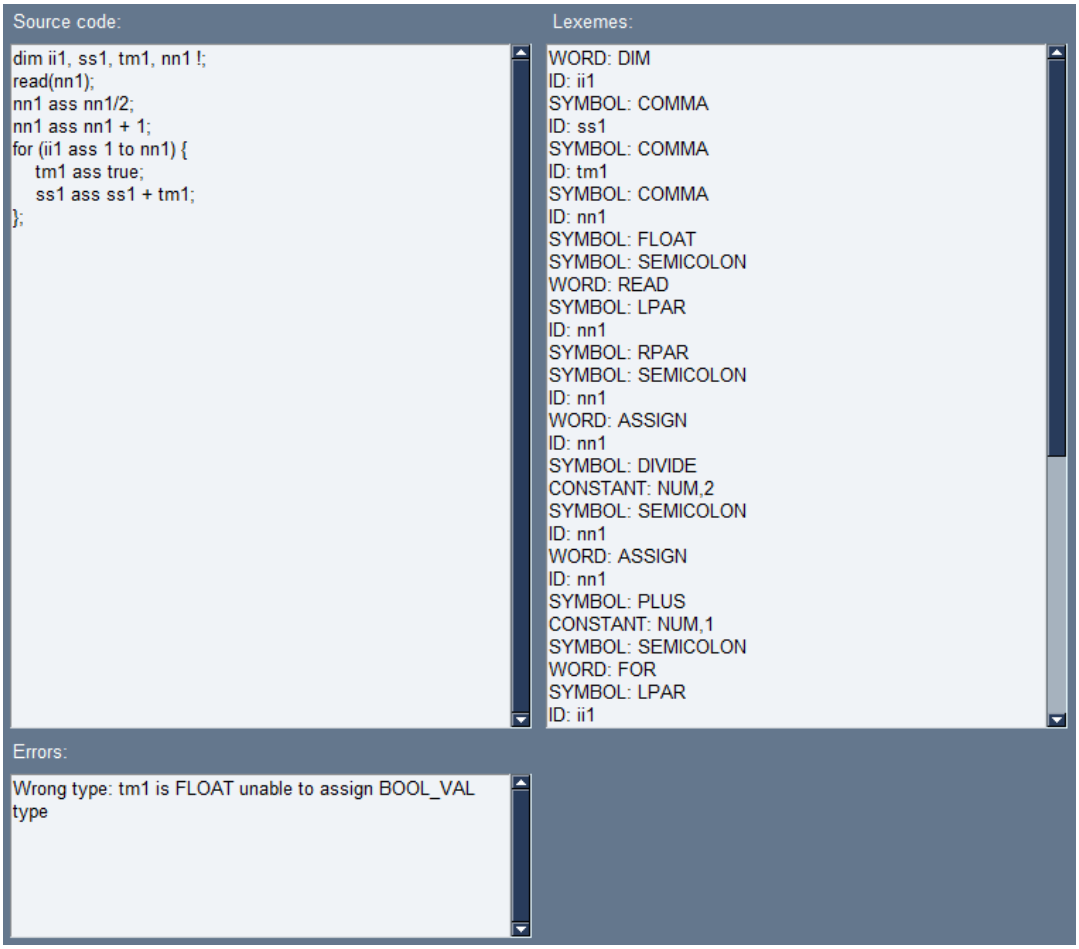

Рисунок 8 – Тест 2 алгоритма семантического анализа

3 Трансляция в ассемблер

3.1 Перевод в польскую инверсную запись

3.1.1 Алгоритм

Для перевода в польскую инверсную запись используется алгоритм, основанный на алгоритме Замельсона и Бауэра, который заключается в предварительном составлении таблицы приоритетов для каждого оператора языка. В такой таблице хранятся стековый и магазинный приоритеты. Код читается слева направо и каждый элемент добавляется либо в результат, либо в магазин. Идентификаторы и константы переписываются в выходную строку ПОЛИЗа. При обнаружении разделителя его сравнительный приоритет P_C сравнивается с магазинным приоритетом P_M разделителя из вершины магазина операций. Если $P_C > P_M$, то разделитель входной строки помещается в магазин (разделитель из исходной строки поступает в магазин и в том случае, когда магазин пуст). Если $P_C \leq P_M$, то символ извлекается из магазина и записывается в выходную строку ПОЛИЗа [2]. Ниже представлена таблица магазинного и стекового приоритетов операторов.

```
PRIORITY = {
        "INT":      [2, 2],
        "FLOAT":    [2, 2],
        "BOOL":     [2, 2],
        "(":        [3, 1],
        ")":        [0, 0],
        "+":        [5, 5],
        "-":        [5, 5],
        ">":        [5, 5],
        "<":        [5, 5],
        ">=":       [5, 5],
        "<=":       [5, 5],
        "=":        [5, 5],
        "!=":       [5, 5],
        "*":        [5, 5],
```

```
        "/":        [5, 5],
        ";":        [0, 0],
        ",":        [3, 2],
        "{":        [1, 0],
        "}":        [0, 0],
        "if":       [1, 2],
        "else":     [1, 0],
        "elseif":   [1, 2],
        "for":      [1, 2],
        "while":    [1, 2],
        "ass":      [2, 1],
        "dim":      [2, 2],
        "and":      [5, 5],
        "or":       [5, 5],
        "not":      [5, 5],
        "read":     [2, 2],
        "output":   [2, 2],
        "to":       [0, 0],
    }
```

Особенности алгоритма:

1. Для обозначения меток используется синтаксическая запись `(имя_метки)`.

2. Для обозначения перехода по метке используется синтаксическая запись `[имя_метки]`.

3. Для обозначения присвоения используется синтаксическая запись `ass`.

4. Цикл for представлен в ПОЛИЗе синтаксической записью `ii1 1 ass (M0) ii1 10 [END0] for ... [M0] (END0)` где `ii1` – счётчик цикла, `1` – начальное значение, `10` – конечное значение, `M0` – метка начала цикла, `END0` – метка конца цикла, троеточие – код, выполняющийся внутри цикла.

5. Для обозначения условных переходов используются синтаксические записи `if`, `elseif`, `else`, `while`, `for`.

6. Для обозначения списка идентификаторов в конструкциях `dim`, `read` и `output` используется оператор «`,`». Он объединяет два предыдущих идентификатора или идентификатор и список в один список.

3.1.2 Реализация перевода в ПОЛИЗ

Метод `convert()` класса RPN реализует перевод заданной последовательности операторов и операндов в польскую инверсную запись. Метод возвращает два массива с элементами ПОЛИЗа. `declare_rpn` – отдельный ПОЛИЗ для декларирования переменных, main_rpn - ПОЛИЗ для основной части программы.

При выполнении перевода используется `if_stack` – стек меток, необходимых для перемещения по конструкциям if-elseif-else, `end_stack` – стек меток, необходимых для прыжка на конец конструкций, заканчивающихся на закрывающую фигурную скобку, `cycle_stack` – стек, позволяющий проще определять какой тип цикла в данный момент преобразуется. Это необходимо, т.к. каждый цикл использует свой способ управления метками.

Все приоритеты операторов хранятся в ассоциативной таблице `PRIORITY`. Каждому оператору соответствует свой массив из двух элементов в котором хранится сравнительный и магазинный приоритет.

3.1.3 Тестирование

Задание. Рассчитать сумму чётных чисел от 1 до *n* с использованием конструкции `do while`.

Тест 1. Программа выводит корректную польскую инверсную запись.

Исходный код программы на модульном языке программирования:

```
dim ii1, ss1, nn1 !;
read(nn1);
do while (ii1 <= nn1) {
    ss1 ass ss1 + ii1;
```

```
        ii1 ass ii1 + 2;
    };
```

Результат работы алгоритма преобразования в ПОЛИЗ представлен на рисунке 9.

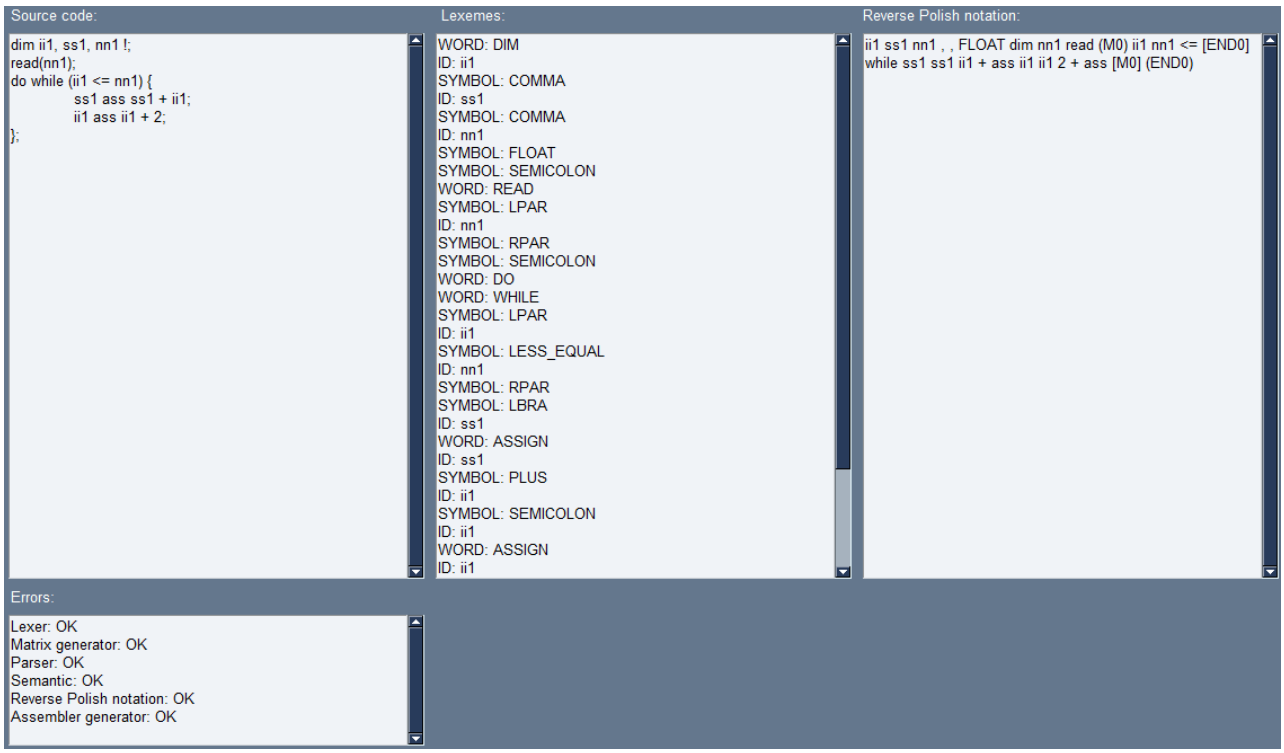

Рисунок 9 – Тест 1 алгоритма преобразования в ПОЛИЗ

3.2 Генерация ассемблерного кода из ПОЛИЗА

3.2.1 Алгоритм

Для перевода в ассемблер NASM используется алгоритм, считывающий поэлементно ПОЛИЗ и генерирующий ассемблерный код в соответствии со встретившимся элементом.

3.2.2 Реализация перевода в ассемблерный код языка NASM

Метод `generate()` класса ASM реализует перевод заданной польской инверсной записи в ассемблерный язык NASM x86. Он принимает на вход 2 части ПОЛИЗа – `declare_rpn` и `main_rpn`. Первая часть используется для

декларирования переменных в сегменте данных, а вторая для реализации алгоритма программы в сегменте кода. Все числа представлены в формате single IEEE-754. Ввод и вывод происходит в том же формате с использованием шестнадцатеричного кодирования. Для конвертации чисел из десятичных чисел модульного языка программирования в нужный формат ассемблерного языка используется библиотека ieee754 языка Python. Ассемблерный код полученный в результате данного алгоритма можно перевести в объектный файл с помощью программы NASM и скомпилировать данный файл в исполняемый посредствам компилятора GCC или же запустить и удостоверится в его работе с помощью программы SASM. Создание исполняемого файла возможно нажатием кнопки «Build EXE» графического интерфейса программы [3].

3.2.3 Тестирование

Задание 1. Вычислить *n*-ое число Фибоначчи.

Исходный код программы на модульном языке программирования

```
dim aa1, aa2, ss1, nn1, ii1 !;
aa1 ass 1;
aa2 ass 1;
nn1 ass 10;
ii1 ass 3;
do while (ii1 <= nn1) {
    ss1 ass aa1 + aa2;
    aa1 ass aa2;
    aa2 ass ss1;
    ii1 ass ii1 + 1;
};
output(ss1);
```

Результат работы алгоритма генерации ассемблерного кода представлен на рисунке 10.

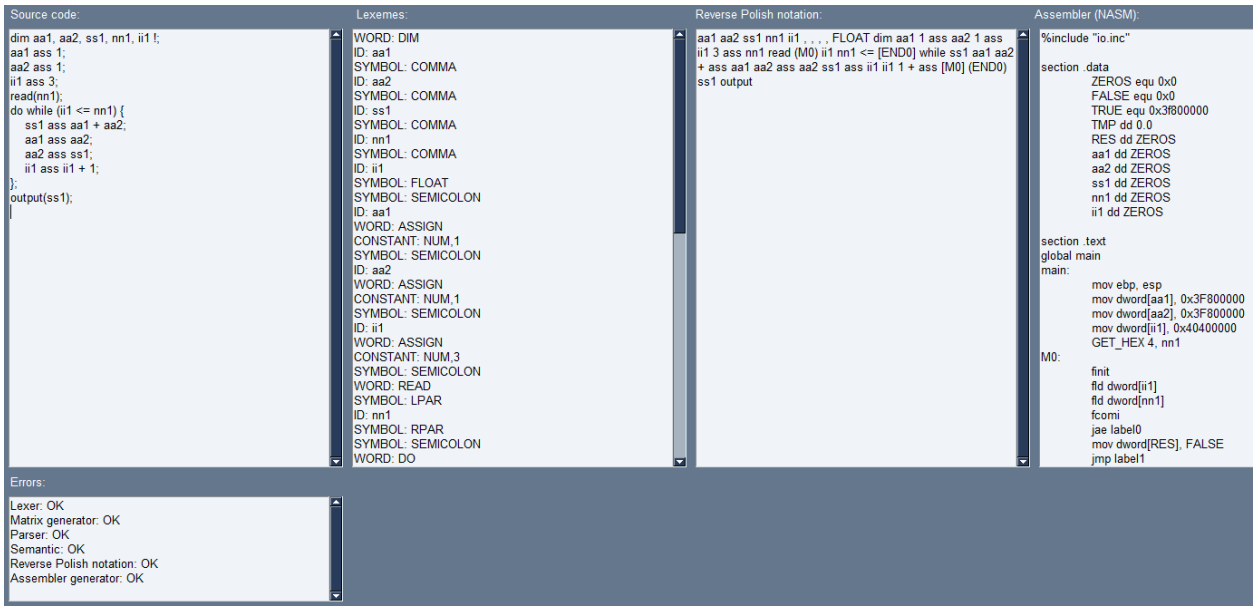

Рисунок 10 – Задание 1 результат работы алгоритма генерации ассемблерного кода

Полученный код на языке ассемблера:

```
%include "io.inc"

section .data
      ZEROS equ 0x0
      FALSE equ 0x0
      TRUE equ 0x3f800000
      TMP dd 0.0
      RES dd ZEROS
      aa1 dd ZEROS
      aa2 dd ZEROS
      ss1 dd ZEROS
      nn1 dd ZEROS
      ii1 dd ZEROS

section .text
global main
main:
      mov ebp, esp
      mov dword[aa1], 0x3F800000
      mov dword[aa2], 0x3F800000
```

```
      mov dword[ii1], 0x40400000
      GET_HEX 4, nn1
M0:
      finit
      fld dword[ii1]
      fld dword[nn1]
      fcomi
      jae label0
      mov dword[RES], FALSE
      jmp label1
label0:
      mov dword[RES], TRUE
label1:
      finit
      fld dword[RES]
      mov dword[TMP], 0x00000000
      fld dword[TMP]
      fcomi
      jz END0
      finit
      fld dword[aa1]
      fld dword[aa2]
      fadd
      fstp dword[RES]
      mov eax, dword[RES]
      mov dword[ss1], eax
      mov eax, dword[aa2]
      mov dword[aa1], eax
      mov eax, dword[ss1]
      mov dword[aa2], eax
      finit
      fld dword[ii1]
      mov dword[TMP], 0x3F800000
      fld dword[TMP]
      fadd
      fstp dword[RES]
      mov eax, dword[RES]
      mov dword[ii1], eax
```

```
        jmp M0
END0:
        PRINT_HEX 4, ss1
NEWLINE
        ret
```

При выполнении ассемблерного кода в среде SASM можно удостовериться, что ассемблирование прошло успешно. В ячейке памяти с именем ss1 хранится результат вычисления n-ого числа Фибоначчи. Значение n вводится пользователем а значение n-ого числа Фибоначчи выводится на экран. Входные данные 0x41200000 соответствует числу 10, а 0x425C0000 соотвествует 55 в кодировке IEEE-754.

Результат работы ассемблерного кода на языке NASM представлен на рисунке 11.

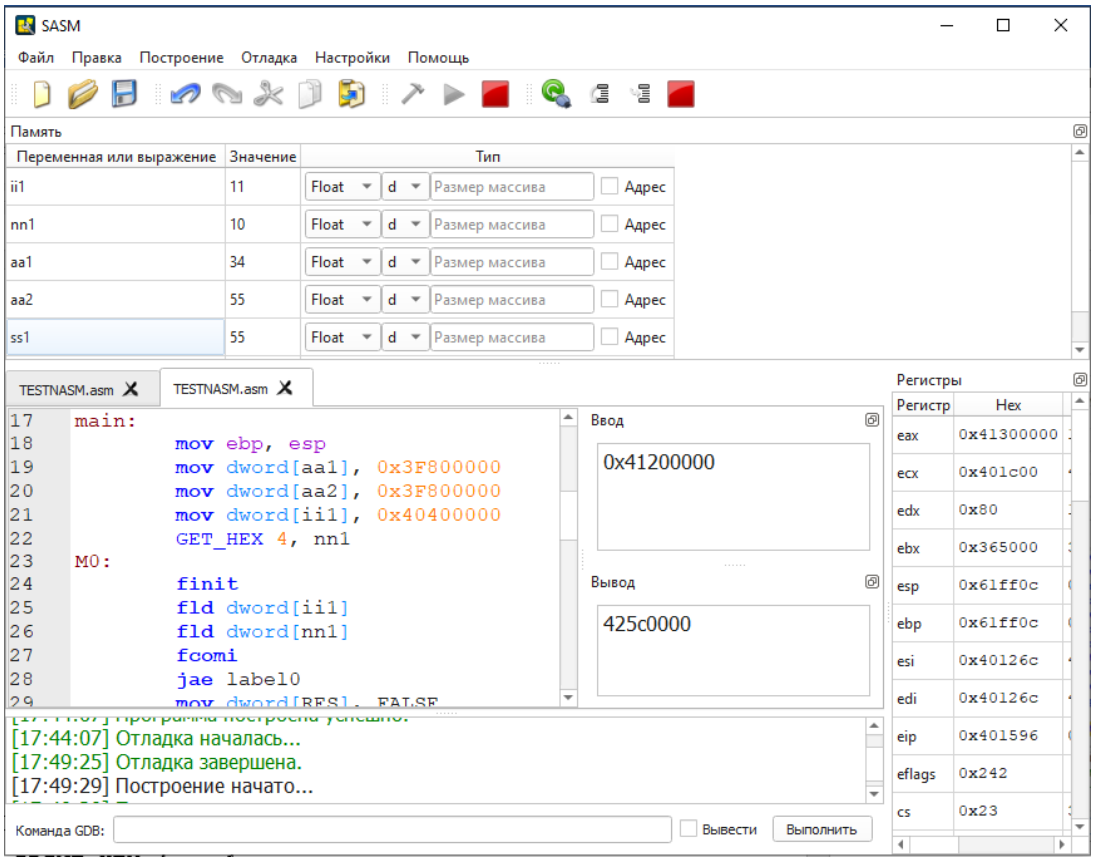

Рисунок 11 – Задание 1 выполнение ассемблерного кода в среде SASM

Задание 2. Выяснить взаимное расположение прямых и найти точку пересечения, если она есть.

Исходный код программы на модульном языке программирования

```
dim kk1, kk2, bb1, bb2, xx1, yy1, tm1 !;
dim sm1, pr1, eq1, eq2 $;
read(kk1, bb1, kk2, bb2);
eq1 ass kk1 = kk2;
eq2 ass bb1 = bb2;
if (eq1 and eq2) {
    sm1 ass true;
} elseif (eq1 = true) {
    pr1 ass true;
} else {
    xx1 ass bb2 - bb1;
    tm1 ass kk1 - kk2;
    xx1 ass xx1 / tm1;
    yy1 ass kk1 * xx1;
    yy1 ass yy1 + bb1;
};
output(xx1, yy1);
```

Результат работы алгоритма генерации ассемблерного кода представлен на рисунке 12.

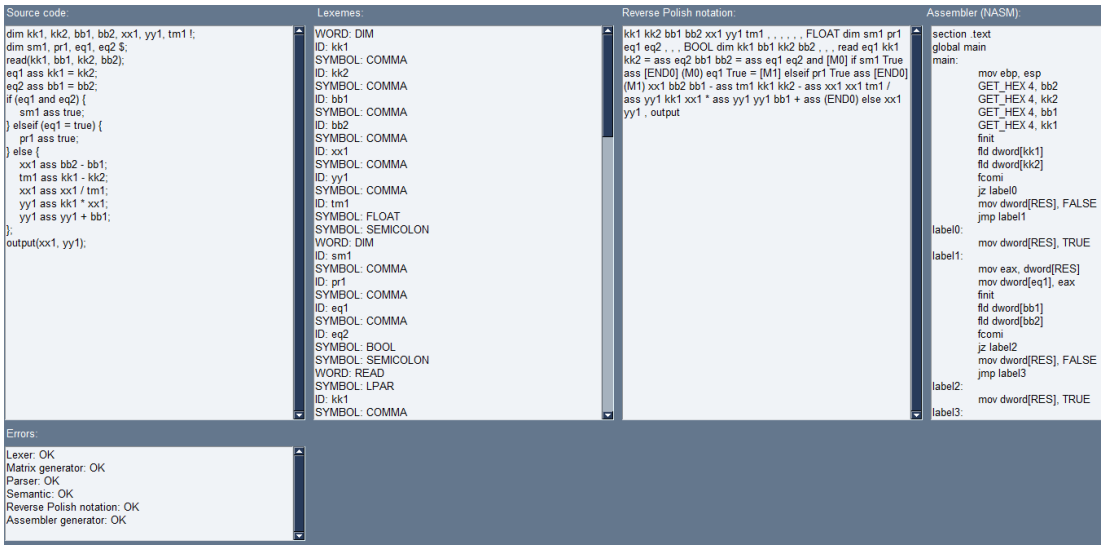

Рисунок 12 – Задание 2 результат работы алгоритма генерации ассемблерного кода

Полученный код на языке ассемблера:

```
%include "io.inc"

section .data
      ZEROS equ 0x0
      FALSE equ 0x0
      TRUE equ 0x3f800000
      TMP dd 0.0
      RES dd ZEROS
      kk1 dd ZEROS
      kk2 dd ZEROS
      bb1 dd ZEROS
      bb2 dd ZEROS
      xx1 dd ZEROS
      yy1 dd ZEROS
      tm1 dd ZEROS
      sm1 dd ZEROS
      pr1 dd ZEROS
      eq1 dd ZEROS
      eq2 dd ZEROS

section .text
global main
main:
      mov ebp, esp
      GET_HEX 4, bb2
      GET_HEX 4, kk2
      GET_HEX 4, bb1
      GET_HEX 4, kk1
      finit
      fld dword[kk1]
      fld dword[kk2]
      fcomi
      jz label0
      mov dword[RES], FALSE
      jmp label1
label0:
```

```
      mov dword[RES], TRUE
label1:
      mov eax, dword[RES]
      mov dword[eq1], eax
      finit
      fld dword[bb1]
      fld dword[bb2]
      fcomi
      jz label2
      mov dword[RES], FALSE
      jmp label3
label2:
      mov dword[RES], TRUE
label3:
      mov eax, dword[RES]
      mov dword[eq2], eax
      mov eax, dword[eq1]
      and eax, dword[eq2]
      mov dword[RES], eax
      finit
      fld dword[RES]
      mov dword[TMP], 0x00000000
      fld dword[TMP]
      fcomi
      jnz label4
      jmp M0
label4:
      mov dword[sm1], TRUE
      jmp END0
M0:
      finit
      fld dword[eq1]
      mov dword[TMP], TRUE
      fld dword[TMP]
      fcomi
      jz label5
      mov dword[RES], FALSE
      jmp label6
```

```
label5:
      mov dword[RES], TRUE
label6:
      finit
      fld dword[RES]
      mov dword[TMP], 0x00000000
      fld dword[TMP]
      fcomi
      jnz label7
      jmp M1
label7:
      mov dword[pr1], TRUE
      jmp END0
M1:
      finit
      fld dword[bb2]
      fld dword[bb1]
      fsub
      fstp dword[RES]
      mov eax, dword[RES]
      mov dword[xx1], eax
      finit
      fld dword[kk1]
      fld dword[kk2]
      fsub
      fstp dword[RES]
      mov eax, dword[RES]
      mov dword[tm1], eax
      finit
      fld dword[xx1]
      fld dword[tm1]
      fdiv
      fstp dword[RES]
      mov eax, dword[RES]
      mov dword[xx1], eax
      finit
      fld dword[kk1]
      fld dword[xx1]
```

```
        fmul
        fstp dword[RES]
        mov eax, dword[RES]
        mov dword[yy1], eax
        finit
        fld dword[yy1]
        fld dword[bb1]
        fadd
        fstp dword[RES]
        mov eax, dword[RES]
        mov dword[yy1], eax
END0:
        PRINT_HEX 4, yy1
NEWLINE
        PRINT_HEX 4, xx1
NEWLINE
        ret
```

Результат работы ассемблерного кода на языке NASM для входных данных k1 = 2, b1 = -1, k2 = -3, b2 = 1 представлен на рисунках 13-14.

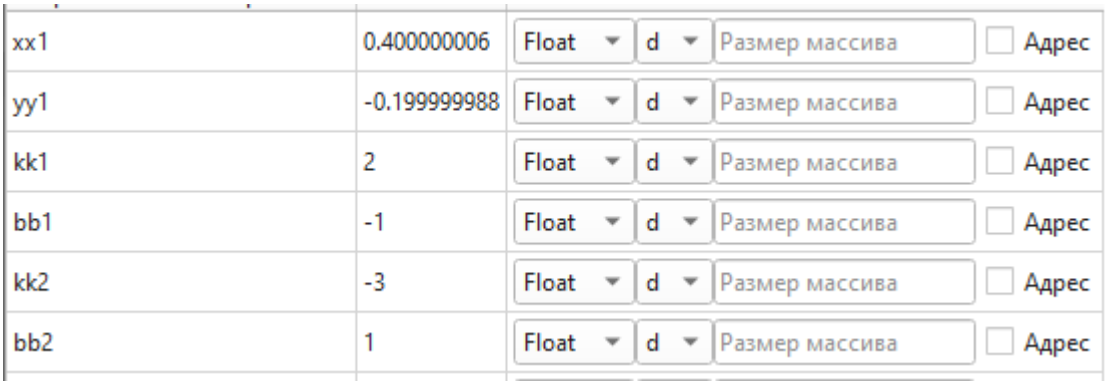

Рисунок 13 – Задание 2 ячейки памяти в результате выполения программы для пересекающихся прямых

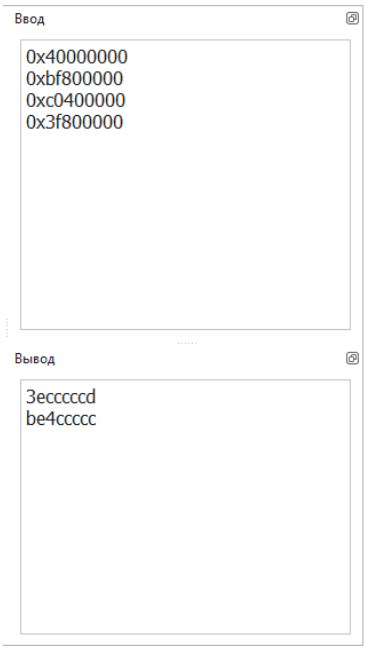

Рисунок 14 – Задание 2 входные и выходные данные программы

0x3ECCCCCD и 0xBE4CCCCC в кодировке IEEE-754 соответственно равны x = 0.4 и y = -0.2.

Результат работы ассемблерного кода на языке NASM для входных данных k1 = 2, b1 = -1, k2 = 2, b2 = 1 представлен на рисунке 15. По результату pr1 = 1 видно, что две прямые параллельны.

Память

Переменная или выражение	Значение	Тип
xx1	0	Float d Размер массива Адрес
yy1	0	Float d Размер массива Адрес
kk1	2	Float d Размер массива Адрес
bb1	-1	Float d Размер массива Адрес
kk2	2	Float d Размер массива Адрес
bb2	1	Float d Размер массива Адрес
sm1	0	Float d Размер массива Адрес
pr1	1	Float d Размер массива Адрес

Рисунок 15 – Задание 2 ячейки памяти в результате выполения программы для параллельных прямых

Результат работы ассемблерного кода на языке NASM для входных данных k1 = 2, b1 = 1, k2 = 2, b2 = 1 представлен на рисунке 16. По результату sm1 = 1 видно, что прямые являются одинаковыми и у них бесконечное количество общих точек.

Переменная или выражение	Значение	Тип
xx1	0	Float d Размер массива Адрес
yy1	0	Float d Размер массива Адрес
kk1	2	Float d Размер массива Адрес
bb1	1	Float d Размер массива Адрес
kk2	2	Float d Размер массива Адрес
bb2	1	Float d Размер массива Адрес
pr1	0	Float d Размер массива Адрес
sm1	1	Float d Размер массива Адрес
Добавить...		Smart d Размер массива Адрес

Рисунок 16 – Задание 2 ячейки памяти в результате выполения программы для одинаковых прямых

Ассемблерный код на языке NASM взят из Задания 2 и входные данные взяты из примера для пересекающихся прямых. Созданные объектный и исполняемый файлы с помощью функции «Build EXE» и демонстрация запуска исполняемого файла представлены на рисунках 17-18.

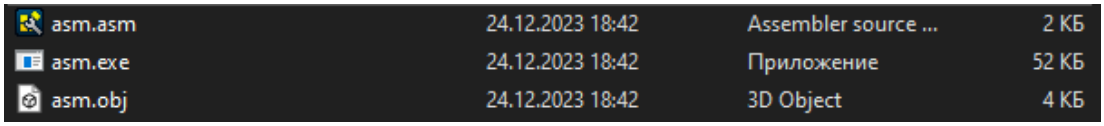

Рисунок 17 – Задание 2 файлы, полученные при построении исполняемого файла

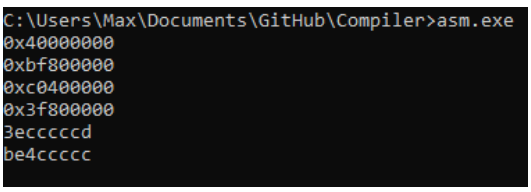

Рисунок 18 – Задание 2 демонстрация работы исполняемого файла

ЗАКЛЮЧЕНИЕ

В процессе разработки программного обеспечения для решения конкретной задачи была изучена специфическая литература по теме работы: литература, посвящённая созданию компиляторов, языкам программирования Python и NASM; разработан алгоритм лексического анализа путём посимвольного разделения исходного кода на лексемы, синтаксического анализа на основе алгоритма сдвиг-свёртки и семантического анализа, реализован алгоритм перевода в ПОЛИЗ на основе алгоритма Замельсона и Бауэра и алгоритм трансляции на ассемблерный язык программирования NASM с возможностью пользовательского ввода-вывода и последующей компиляции в исполняемый файл. Также были изучены библиотеки языка Python, работающие с графическим интерфейсом и форматом чисел IEEE-754.

Программа может быть использована пользователем для компиляции модульного языка программирования в ассемблерный код.

Усовершенствовать данное приложение можно улучшением интерфейса и добавлением новых языковых конструкций в модульный язык программирования.

СПИСОК ИСТОЧНИКОВ

1. Молчанов, А. Ю. Системное программной обеспечение. Лабораторный практикум. / А. Ю. Молчанов – СПб: Питер, 2005. – 284 с.
2. Справка по NASM – Общая информация о языке NASM. [Сайт]. URL: https://www.nasm.us/xdoc/2.16.01/html/nasmdoc3.html (дата обращения 14.12.2023).
3. Справка по Python – Механизм реализации хеш-таблиц. [Сайт]. URL: https://docs.python.org/3/reference/datamodel.html#object.__hash__ (дата обращения 16.09.2023).
4. Малявко, А. А. Формальные языки и компиляторы: учебное пособие для вузов. / А. А. Малявко. – М.: Изд-во Юрайт, 2022. – 429 с.

ПРИЛОЖЕНИЕ А

Текст программы

 # lexer.py

```
decrypt = ['NUM', 'REAL', 'ID', 'if', 'else', 'elseif', 'for', 'to',
'do', 'while', '(', ')', '+', '-', '>', '>=', '<', '<=', '=', '!=',
'*', '/', ';', '.', ',', '{', '}', 'ass', 'dim', 'and', 'or', 'not',
'BOOL_VAL', 'read', 'output', 'INT', 'FLOAT', 'BOOL', 'EOF']

decrypt_to_operators = ['NUM', 'NUM', 'ID', 'if', 'else', 'elseif',
'for', 'to', 'do', 'while', '(', ')', '+', '-', '>', '>=', '<', '<=',
'=', '!=', '*', '/', ';', '.', ',', '{', '}', 'ass', 'dim', 'and',
'or', 'not', 'BOOL', 'read', 'output', 'TYPE', 'TYPE', 'TYPE', '/e/']

decrypt_to_lexemes = ["NUM", "REAL", "ID", "IF", "ELSE", "ELSEIF",
"FOR", "TO", "DO", "WHILE", "LPAR", "RPAR", "PLUS", "MINUS", "GREAT",
"GREAT_EQUAL", "LESS", "LESS_EQUAL", "EQUAL", "NOTEQUAL", "MULTIPLY",
"DIVIDE", "SEMICOLON", "DOT", "COMMA", "LBRA", "RBRA", "ASSIGN",
"DIM", "AND", "OR", "NOT", "BOOL_VAL", "READ", "OUTPUT", "INT",
"FLOAT", "BOOL", "EOF"]

class Lexer:
    (NUM, REAL, ID, IF, ELSE, ELSEIF, FOR, TO, DO, WHILE, LPAR, RPAR,
PLUS, MINUS, GREAT, GREAT_EQUAL, LESS, LESS_EQUAL, EQUAL, NOTEQUAL,
MULTIPLY, DIVIDE, SEMICOLON, DOT, COMMA, LBRA, RBRA, ASSIGN, DIM, AND,
OR, NOT, BOOL_VAL, READ, OUTPUT, INT, FLOAT, BOOL, EOF) = range(39)
    SYMBOLS = {';': SEMICOLON, '+': PLUS, '-': MINUS, '*': MULTIPLY,
'/': DIVIDE, '>': GREAT, '>=': GREAT_EQUAL, '<': LESS, '<=':
LESS_EQUAL, '=': EQUAL, '!=': NOTEQUAL, '.': DOT, ',': COMMA, '(':
LPAR, ')': RPAR, '{': LBRA, '}': RBRA, '%': INT, "!": FLOAT, '$':
BOOL}
    WORDS = {'if': IF, 'else': ELSE, 'elseif': ELSEIF, 'for': FOR,
'to': TO, 'do': DO, 'while': WHILE, 'ass': ASSIGN, 'dim': DIM, 'read':
READ, 'output': OUTPUT, 'and': AND, 'or': OR, 'not': NOT, 'true':
BOOL_VAL, 'false': BOOL_VAL}
    char = ' '
    lines_count = 1
```

```
def __init__(self, input_stream):
    self.symbol = None
    self.value = None
    self.input_stream = input_stream
    self.error = False
    self.error_msg = "Lexer: OK"
def set_error(self, msg):
    self.error = True
    self.error_msg = "Lexer: " + msg
def getc(self):
    self.char = self.input_stream.read(1)
    if self.char == "\n":
        self.lines_count += 1
def next_token(self):
    self.value = None
    self.symbol = None
    while self.symbol is None:
        if len(self.char) == 0:
            self.symbol = Lexer.EOF
        elif self.char.isspace():
            self.getc()
        elif self.char in Lexer.SYMBOLS:
            new_symbol = self.char
            self.getc()
            double_symbol = new_symbol + self.char
            if new_symbol == '-':
                if self.char.isdigit():
                    intval = int(self.char)
                    self.getc()
                    while self.char.isdigit():
                        intval = intval * 10 + int(self.char)
                        self.getc()
                    if self.char == '.':
                        floatval = 0
                        count = 1
                        self.getc()
                        while self.char.isdigit():
                            floatval = floatval + int(self.char) /
```

```
(10 ** count)
                                count += 1
                                self.getc()
                            self.value = -(intval + floatval)
                            self.symbol = Lexer.REAL
                        else:
                            self.value = -intval
                            self.symbol = Lexer.NUM
                    else:
                        self.symbol = Lexer.SYMBOLS[new_symbol]
                elif double_symbol in Lexer.SYMBOLS:
                    self.symbol = Lexer.SYMBOLS[double_symbol]
                    self.getc()
                elif double_symbol == '/*':
                    out_char = ''
                    while out_char != '*/':
                        self.getc()
                        if len(self.char) == 0:
                            self.symbol = Lexer.EOF
                            break
                        out_char += self.char
                        if out_char == '*' or out_char == '*/':
                            continue
                        out_char = ''
                    self.getc()
                else:
                    self.symbol = Lexer.SYMBOLS[new_symbol[0]]
            elif self.char.isdigit():
                intval = 0
                while self.char.isdigit():
                    intval = intval * 10 + int(self.char)
                    self.getc()
                if self.char == '.':
                    floatval = 0
                    count = 1
                    self.getc()
                    while self.char.isdigit():
                        floatval = floatval + int(self.char) / (10 **
```

```
count)
                        count += 1
                        self.getc()
                    self.value = intval + floatval
                    self.symbol = Lexer.REAL
                else:
                    self.value = intval
                    self.symbol = Lexer.NUM
            elif 'a' <= self.char <= 'z':
                identifier = ''
                while 'a' <= self.char <= 'z':
                    identifier += self.char.lower()
                    self.getc()
                if identifier in Lexer.WORDS:
                    self.symbol = Lexer.WORDS[identifier]
                    if identifier == "false":
                        self.value = False
                    elif identifier == "true":
                        self.value = True
                elif len(identifier) == 2 and self.char.isdigit():
                    identifier += self.char
                    self.getc()
                    while self.char.isdigit():
                        identifier += self.char
                        self.getc()
                    self.symbol = Lexer.ID
                    self.value = identifier
                else:
                    self.set_error('Unknown identifier "' + identifier
+ '" in line ' + str(self.lines_count))
                    break
            else:
                self.set_error('Unexpected symbol "' + self.char + '"
in line ' + str(self.lines_count))
                break
```

 # matrix.py

```
from enum import Enum
```

```
class ORDER(Enum):
    PRECEDED = 1
    FOLLOWS = 2
    EQUALS = 3
class LastSymbols:
    def __init__(self):
        self.left = {}
        self.right = {}
    def __eq__(self, other):
        if isinstance(other, LastSymbols):
            return self.left == other.left and self.right ==
other.right
        return NotImplemented
class MatrixGenerator:
    SYMBOLS = []
    WORDS = []
    T = []
    def __init__(self, input_stream):
        self.input_stream = input_stream
        self.operator_matrix = {}
        self.error = False
        self.error_msg = "Matrix generator: OK"
        self.rules = {}
        self.symbols_lr = LastSymbols()
        self.t_lr = LastSymbols()
    def set_error(self, msg):
        self.error = True
        self.error_msg = "Matrix generator: " + msg
    def print_matrix(self):
        print(" " * 6, end="")
        for x in [*MatrixGenerator.T, "/e/"]:
            print("{0:<6}".format(x), end="")
        print()
        for i in [*MatrixGenerator.T, "/b/"]:
            print("{0:<6}".format(i), end="")
            for j in [*MatrixGenerator.T, "/e/"]:
                if self.operator_matrix[i].get(j) is None:
                    print("." + " " * 5, end="")
```

```
                elif self.operator_matrix[i][j] == ORDER.PRECEDED:
                    print("<." + " " * 4, end="")
                elif self.operator_matrix[i][j] == ORDER.EQUALS:
                    print("=." + " " * 4, end="")
                elif self.operator_matrix[i][j] == ORDER.FOLLOWS:
                    print(".>" + " " * 4, end="")
            print()
    def generate(self):
        rule_names = self.input_stream.readline().split()
        MatrixGenerator.T = self.input_stream.readline().split()
        for name in rule_names:
            self.rules[name] = []
            self.symbols_lr.left[name] = []
            self.symbols_lr.right[name] = []
            self.t_lr.left[name] = []
            self.t_lr.right[name] = []
        for symbol in ["/b/", *MatrixGenerator.T, "/e/"]:
            self.operator_matrix[symbol] = {}
        lines = self.input_stream.read().split('\n')
        for line in lines:
            last_token_index = 0
            tokens = line.split()
            if tokens[1] == ":" and tokens[0] in rule_names and
len(tokens) > 2:
                rule_name = tokens[0]
                tokens = tokens[2:]
                for i in range(len(tokens)):
                    if tokens[i] == "|":

self.rules[rule_name].append(tokens[last_token_index:i])
                        last_token_index = i+1
                    elif tokens[i] not in MatrixGenerator.T and
tokens[i] not in rule_names:
                        self.set_error('unknown symbol "' + tokens[i]
+ '"')
                        return self.error

self.rules[rule_name].append(tokens[last_token_index:])
```

```
            else:
                self.set_error("wrong file formatting")
                return self.error
        for rule_name in self.rules.keys():  # начальное заполнение
крайних левых, правых
            for arr in self.rules[rule_name]:
                first_t = None
                last_t = None
                first_symbol = None
                last_symbol = None
                for item in arr:
                    if item in MatrixGenerator.T:
                        if first_t is None:
                            first_t = item
                            last_t = item
                        else:
                            last_t = item
                    if first_symbol is None:
                        first_symbol = item
                        last_symbol = item
                    else:
                        last_symbol = item
                if last_t is not None:
                    if first_t not in self.t_lr.left[rule_name]:
                        self.t_lr.left[rule_name].append(first_t)
                    if last_t not in self.t_lr.right[rule_name]:
                        self.t_lr.right[rule_name].append(last_t)
                if last_symbol is not None:
                    if first_symbol not in
self.symbols_lr.left[rule_name]:
self.symbols_lr.left[rule_name].append(first_symbol)
                    if last_symbol not in
self.symbols_lr.right[rule_name]:
self.symbols_lr.right[rule_name].append(last_symbol)
        changed = True
        while changed:  # алгоритм для всех симв
            changed = False
            for rule_name in self.rules.keys():
```

```
                for item in self.symbols_lr.left[rule_name]:
                    if item in self.rules.keys():
                        for i in self.symbols_lr.left[item]:
                            if i not in
self.symbols_lr.left[rule_name]:
self.symbols_lr.left[rule_name].append(i)
                                changed = True
                for item in self.symbols_lr.right[rule_name]:
                    if item in self.rules.keys():
                        for i in self.symbols_lr.right[item]:
                            if i not in
self.symbols_lr.right[rule_name]:
self.symbols_lr.right[rule_name].append(i)
                                changed = True
        for rule_name in self.rules.keys():  # алгоритм для терм симв
            for item in self.symbols_lr.left[rule_name]:
                if item not in MatrixGenerator.T:
                    for i in self.t_lr.left[item]:
                        if i not in self.t_lr.left[rule_name] and i in
MatrixGenerator.T:
                            self.t_lr.left[rule_name].append(i)
            for item in self.symbols_lr.right[rule_name]:
                if item not in MatrixGenerator.T:
                    for i in self.t_lr.right[item]:
                        if i not in self.t_lr.right[rule_name] and i
in MatrixGenerator.T:
                            self.t_lr.right[rule_name].append(i)
        for symbol in MatrixGenerator.T:
            for rule in self.rules.values():
                for item in rule:
                    for i in range(len(item)):
                        if symbol == item[i]:
                            if i < len(item)-1 and item[i+1] in
MatrixGenerator.T:  # x ai b y
                                if
self.operator_matrix[symbol].get(item[i+1]) is not None and
self.operator_matrix[symbol][item[i+1]] != ORDER.EQUALS:
```

```
                                    print("ОШИБКА", item, item[i:i+2],
"x ai b y OLD:", self.operator_matrix[symbol][item[i+1]])
                                # print(item[i:i + 2], "x ai b y")

self.operator_matrix[symbol][item[i+1]] = ORDER.EQUALS
                            elif i < len(item)-2 and item[i+1] not in
MatrixGenerator.T and item[i+2] in MatrixGenerator.T:  # x ai U b y
                                if
self.operator_matrix[symbol].get(item[i+2]) is not None and
self.operator_matrix[symbol][item[i+2]] != ORDER.EQUALS:
                                    print("ОШИБКА", item, item[i:i+3],
"x ai U b y")
                                # print(item[i:i + 3], "x ai U b y")

self.operator_matrix[symbol][item[i+2]] = ORDER.EQUALS
                            if i < len(item)-1 and item[i+1] not in
MatrixGenerator.T:  # x ai U y
                                for L in self.t_lr.left[item[i+1]]:
                                    if L in MatrixGenerator.T:
                                        if
self.operator_matrix[symbol].get(L) is not None and
self.operator_matrix[symbol][L] != ORDER.PRECEDED:
                                            print("ОШИБКА", item, L,
item[i:i + 2], "x ai U y")
                                        # print(L, item[i:i + 2], "x
ai U y")

self.operator_matrix[symbol][L] = ORDER.PRECEDED
                                        continue


                            if i > 0 and item[i-1] not in
MatrixGenerator.T:  # x U ai y
                                for R in self.t_lr.right[item[i-1]]:
                                    if R in MatrixGenerator.T:
                                        if
self.operator_matrix[R].get(symbol) is not None and
self.operator_matrix[R][symbol] != ORDER.FOLLOWS:
```

```
                                        print("ОШИБКА", item, R,
item[i - 1:i + 1], "x U ai y")

self.operator_matrix[R][symbol] = ORDER.FOLLOWS
                                        continue

        for s in self.t_lr.left[rule_names[0]]:
            self.operator_matrix["/b/"][s] = ORDER.PRECEDED

        for s in self.t_lr.right[rule_names[0]]:
            self.operator_matrix[s]["/e/"] = ORDER.FOLLOWS

        self.print_matrix()
        return self.operator_matrix
```

 # parser.py

```
import matrix
class Parser:
    def __init__(self, op_matrix, operators, rules):
        self.operators = operators
        self.t = ['/b/', *operators]
        self.operator_matrix = op_matrix
        self.error = False
        self.error_msg = "Parser: OK"
        self.stack = []
        self.rules: dict = rules
    def set_error(self, msg):
        self.error = True
        self.error_msg = "Parser: " + msg
    def print_matrix(self):
        pass
    def get_t(self, n=1):
        if n < 1:
            n = 1
        for item in self.stack[::-1]:
            if item in self.t:
                n -= 1
                if n == 0:
```

```
                    return item
        return "ERROR"
    def check(self):
        self.stack.append('/b/')
        i = 0
        while True:
            if self.get_t() == '/b/' and self.operators[i] == '/e/':
                break
            if self.get_t() not in self.operator_matrix.keys() or
self.operators[i] not in self.operator_matrix[self.get_t()].keys():
                self.set_error("unknown construction '" + self.get_t()
+ " " + self.operators[i] + "' in " + " ".join(self.operators[i-
2:i+1]))
                break
            if self.operator_matrix[self.get_t()][self.operators[i]]
== matrix.ORDER.EQUALS or
self.operator_matrix[self.get_t()][self.operators[i]] ==
matrix.ORDER.PRECEDED:
                self.stack.append(self.operators[i])
                i += 1
            elif self.operator_matrix[self.get_t()][self.operators[i]]
== matrix.ORDER.FOLLOWS:
                rule = []
                if self.operator_matrix[self.get_t(2)][self.get_t()]
== matrix.ORDER.PRECEDED:
                    count = 1
                    while True:
                        if self.stack[-1] in self.t:
                            if count == 0:
                                break
                            count -= 1
                        rule.append(self.stack.pop())
                    find = False
                    rule.reverse()
                    for name in self.rules.keys():
                        for r in self.rules[name]:
                            if rule == r:
                                find = True
```

```
                                self.stack.append("E")
                                break
                        if find:
                            break
                    if not find:
                        self.set_error("Unable to locate rule " + "
".join(rule))
                        break
                elif self.operator_matrix[self.get_t(2)][self.get_t()]
== matrix.ORDER.EQUALS:
                    count = 2
                    while
self.operator_matrix[self.get_t(count+1)][self.get_t(count)] ==
matrix.ORDER.EQUALS:
                        count += 1
                    while True:
                        if self.stack[-1] in self.t:
                            if count == 0:
                                break
                            count -= 1
                        rule.append(self.stack.pop())
                    find = False
                    rule.reverse()
                    # print("RULE:", rule)
                    for name in self.rules.keys():
                        for r in self.rules[name]:
                            if rule == r:
                                find = True
                                self.stack.append("E")
                                break
                        if find:
                            break
                    if not find:
                        self.set_error("Unable to locate rule " + "
".join(rule))
                        break
```

 # semantic.py

```
import lexer
class Variable:
    def __init__(self, type):
        self.type = type
class Operator:
    def __init__(self, type, name, value=None):
        self.type = type
        self.name = name
        self.value = value
class Semantic:
    RETURN_BOOL = [">", ">=", "<", "<=", "=", "!=", "and", "or",
"not"]
    RETURN_NUM = ["+", "-", "*", "/"]
    def __init__(self, operators):
        self.operators: list[Operator] = operators
        self.error = False
        self.error_msg = "Semantic: OK"
        self.variables: dict[Variable] = {}
    def set_error(self, msg):
        self.error = True
        self.error_msg = "Semantic: " + msg
    def check(self):
        i = 0
        while i < len(self.operators):
            if self.operators[i].type == lexer.Lexer.DIM:
                j = i + 1
                while self.operators[j].name != "TYPE":
                    j += 1
                vars_type = lexer.decrypt[self.operators[j].type]
                for x in range(i + 1, j + 1, 2):
                    self.variables[self.operators[x].value] =
Variable(vars_type)
            elif self.operators[i].type == lexer.Lexer.ID and
self.operators[i + 1].type == lexer.Lexer.ASSIGN:
                if self.variables.get(self.operators[i].value) is
None:
                    self.error = True
```

```
                    self.error_msg = "Undeclared variable " +
self.operators[i].value
                id_type = self.variables[self.operators[i].value].type
                const_type = lexer.decrypt[self.operators[i + 2].type]
                operation_return = lexer.decrypt[self.operators[i +
3].type]
                if operation_return == ";" or operation_return ==
"to":
                    if const_type == "ID" and
self.variables[self.operators[i].value].type != id_type:
                        print("ID SEMANTIC")
                        self.error = True
                        if self.variables.get(self.operators[i +
2].value) is None:
                            self.error_msg = "Undeclared variable " +
self.operators[i+2].value
                            break
                        self.error_msg = ("Wrong type: " +
self.operators[i].value + " is " +
                                          id_type + " unable to assign
" +

self.variables[self.operators[i + 2].value].type + " type")
                        break
                    elif const_type != "ID" and not ((id_type == "INT"
or id_type == "FLOAT") and (const_type == "NUM"
                         or const_type == "REAL") or (id_type ==
"BOOL" and const_type == "BOOL_VAL")):
                        print("CONST SEMANTIC")
                        self.error = True
                        self.error_msg = ("Wrong type: " +
self.operators[i].value + " is " +
                                          id_type + " unable to assign
" +
                                          const_type + " type")
                        break
                elif ((id_type == "INT" or id_type == "FLOAT") and
operation_return not in
```

```
                        self.RETURN_NUM) or id_type == "BOOL" and
operation_return not in self.RETURN_BOOL:
                    print(operation_return)
                    print("OPERATOR SEMANTIC")
                    self.error = True
                    if operation_return in self.RETURN_NUM:
                        self.error_msg = ("Wrong type: " +
self.operators[i].value + " is " + id_type
                                          + " unable to assign number
type")
                    else:
                        self.error_msg = ("Wrong type: " +
self.operators[i].value + " is " + id_type
                                          + " unable to assign BOOL
type")
                    break
            elif self.operators[i].type == lexer.Lexer.ID:
                if self.variables.get(self.operators[i].value) is
None:
                    self.error = True
                    self.error_msg = "Undeclared variable " +
self.operators[i].value
            i += 1
```

 # RPN.py

```
class PRN:
    OPERANDS = ["ID", "NUM", "REAL", "BOOL_VAL"]
    UNARY = ["NOT"]
    IGNORED = ["EOF", "do"]
    UNPRINTABLE = [";", "{", "(", "}", ")", "to"]
    PRIORITY = {
        "INT":      [2, 2],
        "FLOAT":    [2, 2],
        "BOOL":     [2, 2],
        "(":        [3, 1],
        ")":        [0, 0],
        "+":        [5, 5],
        "-":        [5, 5],
```

```
        ">":        [5, 5],
        "<":        [5, 5],
        ">=":       [5, 5],
        "<=":       [5, 5],
        "=":        [5, 5],
        "!=":       [5, 5],
        "*":        [5, 5],
        "/":        [5, 5],
        ";":        [0, 0],
        ",":        [3, 2],
        "{":        [1, 0],
        "}":        [0, 0],
        "if":       [1, 2],
        "else":     [1, 0],
        "elseif":   [1, 2],
        "for":      [1, 2],
        "while":    [1, 2],
        "ass":      [2, 1],
        "dim":      [2, 2],
        "and":      [5, 5],
        "or":       [5, 5],
        "not":      [5, 5],
        "read":     [2, 2],
        "output":   [2, 2],
        "to":       [0, 0],
    }
    def __init__(self, operators):
        self.operators = operators
        self.error = False
        self.error_msg = "Reverse Polish notation: OK"
        self.stack = []
        self.end_stack = []
        self.if_stack = []
        self.if_count = 0
        self.end_count = 0
        self.target_counter = ""
        self.cycle_stack = []
    def set_error(self, msg):
```

```
        self.error = True
        self.error_msg = "Reverse Polish notation: " + msg
    def convert(self):
        res = []
        declare_res = []
        is_declare = False
        for i in range(len(self.operators)):
            print("OPERATOR:", self.operators[i][0])
            if self.operators[i][0] in self.OPERANDS:
                if is_declare:
                    declare_res.append(self.operators[i][1])
                else:
                    res.append(self.operators[i][1])
                print("OPERAND")
            else:
                if self.operators[i][0] == "dim":
                    is_declare = True
                if self.operators[i][0] == "for":
                    self.target_counter = self.operators[i+2][1]
                if self.operators[i][0] == "if" or
self.operators[i][0] == "for" or self.operators[i][0] == "while":
                    self.end_stack.append("END" + str(self.end_count))
                    self.end_count += 1
                    self.cycle_stack.append(self.operators[i][0])
                if self.operators[i][0] == "do":
                    self.if_stack.append("M" + str(self.if_count))
                    self.if_count += 1
                    res.append("(" + self.if_stack[-1] + ")")
                if self.operators[i][0] in self.IGNORED:
                    print("CONTINUE WITH: " + self.operators[i][0])
                    continue
                if len(self.stack) == 0 or self.PRIORITY[self.stack[-
1]][1] < self.PRIORITY[self.operators[i][0]][0]:
                    print("STACK:", len(self.stack))
                    self.stack.append(self.operators[i][0])
                else:
                    print("CYCLE:", self.cycle_stack)
                    stack_symbol = ""
```

```
                    if self.operators[i][0] == ")":
                        while stack_symbol != "(":
                            stack_symbol = self.stack.pop()
                            if stack_symbol not in self.UNPRINTABLE:
                                if stack_symbol == "elseif":
                                    stack_symbol = "if"
                                res.append(stack_symbol)
                        stack_symbol = self.stack.pop()
                        if stack_symbol == "if" or stack_symbol ==
"else" or stack_symbol == "elseif":  # else????
                            self.if_stack.append("M" +
str(self.if_count))
                            self.if_count += 1
                            res.append("[" + self.if_stack[-1] + "]")
                        if stack_symbol == "for" or stack_symbol ==
"while":
                            res.append("[" + self.end_stack[-1] + "]")
                        if stack_symbol not in self.UNPRINTABLE:
                            res.append(stack_symbol)
                    if self.operators[i][0] == "}":
                        while stack_symbol != "{":
                            stack_symbol = self.stack.pop()
                            if stack_symbol not in self.UNPRINTABLE:
                                if stack_symbol == "elseif":
                                    stack_symbol = "if"
                                res.append(stack_symbol)
                        if self.operators[i+1][0] == ";":
                            cycle_type = self.cycle_stack.pop()
                            if cycle_type == "for":
                                res.append(self.target_counter)
                                res.append(self.target_counter)
                                res.append("1")
                                res.append("+")
                                res.append("ass")
                                res.append("[" + self.if_stack.pop() +
"]")
                                res.append("(" + self.end_stack.pop()
+ ")")
```

```
                            if cycle_type == "while":
                                res.append("[" + self.if_stack.pop() +
"]")
                                res.append("(" + self.end_stack.pop()
+ ")")
                            if cycle_type == "if":
                                res.append("(" + self.end_stack.pop()
+ ")")
                                if len(self.stack) == 0 or
self.stack[-1] != "else":
                                    res.append("(" +
self.if_stack.pop() + ")")
                        else:
                            # stack_symbol = self.stack.pop()
                            # res.append(stack_symbol)
                            res.append("[" + self.end_stack[-1] + "]")
                            res.append("(" + self.if_stack.pop() +
")")
                    if self.operators[i][0] == ";":
                        while stack_symbol != "{" and len(self.stack)
!= 0:
                            stack_symbol = self.stack.pop()
                            if stack_symbol not in self.UNPRINTABLE:
                                if stack_symbol == "elseif":
                                    stack_symbol = "if"
                                if is_declare:
                                    declare_res.append(stack_symbol)
                                else:
                                    res.append(stack_symbol)
                        is_declare = False
                        if stack_symbol == "{":
                            self.stack.append(stack_symbol)
                    if self.operators[i][0] == "INT" or
self.operators[i][0] == "FLOAT" or self.operators[i][0] == "BOOL":
                        stack_symbol = self.stack.pop()
                        while stack_symbol == ",":
                            declare_res.append(stack_symbol)
                            stack_symbol = self.stack.pop()
```

```
                        self.stack.append(stack_symbol)
                        declare_res.append(self.operators[i][0])
                    if self.operators[i][0] == "to":
                        while stack_symbol != "(":
                            stack_symbol = self.stack.pop()
                            if stack_symbol not in self.UNPRINTABLE:
                                if stack_symbol == "elseif":
                                    stack_symbol = "if"
                                res.append(stack_symbol)
                        self.stack.append(stack_symbol)
                        self.if_stack.append("M" + str(self.if_count))
                        self.if_count += 1
                        res.append("(" + self.if_stack[-1] + ")")
                        res.append(self.target_counter)
            print(res)
            print(declare_res)
            print("---")
        if (len(self.stack) == 1 and self.stack[-1] != ";") and
len(self.stack) > 0:
            print("STACK ERROR:", self.stack)
            self.set_error("Stack is not empty")
        return declare_res, res
```

 # ASM.py

```
from ieee754 import single
class ASM:
    TYPES = ["INT", "BOOL", "FLOAT"]
    OPERATORS = ["+", "-", "*", "/", ">", ">=", "<", "<=", "=", "!=",
",", "if", "else", "elseif", "for", "while", "ass", "read", "output",
"and", "or", "not"]
    def __init__(self, declare_rpn, main_rpn):
        self.error = False
        self.error_msg = "Assembler generator: OK"
        self.declare_rpn = declare_rpn
        self.main_rpn = main_rpn
        self.variables = []
        self.var_names = []
        self.stack = []
```

```
        self.calced = False
        self.labels_count = 0
    def set_error(self, msg):
        self.error = True
        self.error_msg = "Assembler generator: " + msg
    def two_operands_calc(self):
        res = ""
        second = self.stack.pop()
        first = self.stack.pop()
        res += "\tfinit\n"
        if first in self.var_names:
            res += "\tfld dword[" + first + "]\n"
        else:
            if first == "True":
                res += "\tmov dword[TMP], TRUE\n\tfld dword[TMP]\n"
            elif first == "False":
                res += "\tmov dword[TMP], FALSE\n\tfld dword[TMP]\n"
            else:
                res += "\tmov dword[TMP], 0x" + single(first).hex()[0]
+ "\n\tfld dword[TMP]\n"
        if second in self.var_names:
            res += "\tfld dword[" + second + "]\n"
        else:
            if second == "True":
                res += "\tmov dword[TMP], TRUE\n\tfld dword[TMP]\n"
            elif second == "False":
                res += "\tmov dword[TMP], FALSE\n\tfld dword[TMP]\n"
            else:
                res += "\tmov dword[TMP], 0x" +
single(second).hex()[0] + "\n\tfld dword[TMP]\n"
        return res
    def generate(self):
        res = ('%include "io.inc"\n\n'
               'section .data\n'
               '\tZEROS equ 0x0\n'
               '\tFALSE equ 0x0\n'
               '\tTRUE equ 0x3f800000\n'
               '\tTMP dd 0.0\n'
```

```
                '\tRES dd ZEROS\n')
        for el in self.declare_rpn:
            if el == "," or el == "dim":
                continue
            elif el in self.TYPES:
                for i in range(len(self.variables)):
                    if self.variables[i][1] is None:
                        self.variables[i][1] = el
                        res += "\t" + self.variables[i][0] + " dd
ZEROS\n"
            else:
                self.variables.append([el, None])
                self.var_names.append(el)
        res += "\nsection .text\nglobal main\nmain:\n\tmov ebp, esp\n"
        for i in range(len(self.main_rpn)):
            el = str(self.main_rpn[i])
            if el == "+":
                res += self.two_operands_calc()
                res += "\tfadd\n"
                self.calced += 1
                res += "\tfstp dword[RES]\n"
            elif el == "-":
                res += self.two_operands_calc()
                res += "\tfsub\n"
                self.calced += 1
                res += "\tfstp dword[RES]\n"
            elif el == "*":
                res += self.two_operands_calc()
                res += "\tfmul\n"
                self.calced += 1
                res += "\tfstp dword[RES]\n"
            elif el == "/":
                res += self.two_operands_calc()
                res += "\tfdiv\n"
                self.calced += 1
                res += "\tfstp dword[RES]\n"
            elif el == "ass":
                if self.calced:
```

```
                    res += "\tmov eax, dword[RES]\n"
                    res += "\tmov dword["+self.stack.pop()+"], eax\n"
                    self.calced = False
                else:
                    second = self.stack.pop()
                    first = self.stack.pop()
                    if second in self.var_names:
                        res += "\tmov eax, dword[" + second + "]\n"
                        res += "\tmov dword["+first+"], eax\n"
                    elif second == "False":
                        res += "\tmov dword["+first+"], FALSE\n"
                    elif second == "True":
                        res += "\tmov dword["+first+"], TRUE\n"
                    else:
                        res += "\tmov dword["+first+"],
0x"+single(second).hex()[0]+"\n"

            elif el == ">":
                res += self.two_operands_calc()
                res += "\tfcomi\n"
                res += "\tjb label"+str(self.labels_count)+"\n"
                self.labels_count += 1
                res += "\tmov dword[RES], FALSE\n"
                res += "\tjmp label"+str(self.labels_count)+"\n"
                self.labels_count += 1
                res += "label"+str(self.labels_count-2)+":\n"
                res += "\tmov dword[RES], TRUE\n"
                res += "label"+str(self.labels_count-1)+":\n"
                self.calced = True
            elif el == ">=":
                res += self.two_operands_calc()
                res += "\tfcomi\n"
                res += "\tjbe label"+str(self.labels_count)+"\n"
                self.labels_count += 1
                res += "\tmov dword[RES], FALSE\n"
                res += "\tjmp label"+str(self.labels_count)+"\n"
                self.labels_count += 1
                res += "label"+str(self.labels_count-2)+":\n"
```

```
            res += "\tmov dword[RES], TRUE\n"
            res += "label"+str(self.labels_count-1)+":\n"
            self.calced = True
        elif el == "=":
            res += self.two_operands_calc()
            res += "\tfcomi\n"
            res += "\tjz label"+str(self.labels_count)+"\n"
            self.labels_count += 1
            res += "\tmov dword[RES], FALSE\n"
            res += "\tjmp label"+str(self.labels_count)+"\n"
            self.labels_count += 1
            res += "label"+str(self.labels_count-2)+":\n"
            res += "\tmov dword[RES], TRUE\n"
            res += "label"+str(self.labels_count-1)+":\n"
            self.calced = True
        elif el == "<":
            res += self.two_operands_calc()
            res += "\tfcomi\n"
            res += "\tja label"+str(self.labels_count)+"\n"
            self.labels_count += 1
            res += "\tmov dword[RES], FALSE\n"
            res += "\tjmp label"+str(self.labels_count)+"\n"
            self.labels_count += 1
            res += "label"+str(self.labels_count-2)+":\n"
            res += "\tmov dword[RES], TRUE\n"
            res += "label"+str(self.labels_count-1)+":\n"
            self.calced = True
        elif el == "<=":
            res += self.two_operands_calc()
            res += "\tfcomi\n"
            res += "\tjae label"+str(self.labels_count)+"\n"
            self.labels_count += 1
            res += "\tmov dword[RES], FALSE\n"
            res += "\tjmp label"+str(self.labels_count)+"\n"
            self.labels_count += 1
            res += "label"+str(self.labels_count-2)+":\n"
            res += "\tmov dword[RES], TRUE\n"
            res += "label"+str(self.labels_count-1)+":\n"
```

```
                self.calced = True
            elif el == "!=":
                res += self.two_operands_calc()
                res += "\tfcomi\n"
                res += "\tjnz label"+str(self.labels_count)+"\n"
                self.labels_count += 1
                res += "\tmov dword[RES], FALSE\n"
                res += "\tjmp label"+str(self.labels_count)+"\n"
                self.labels_count += 1
                res += "label"+str(self.labels_count-2)+":\n"
                res += "\tmov dword[RES], TRUE\n"
                res += "label"+str(self.labels_count-1)+":\n"
                self.calced = True
            elif el == "and":
                second = self.stack.pop()
                first = self.stack.pop()
                if first in self.var_names:
                    res += "\tmov eax, dword[" + first + "]\n"
                else:
                    res += "\tmov eax, 0x" + single(first).hex()[0] +
"\n"
                if second in self.var_names:
                    res += "\tand eax, dword[" + second + "]\n"
                else:
                    res += "\tand eax, 0x" + single(second).hex()[0] +
"\n"
                res += "\tmov dword[RES], eax\n"
                self.calced = True
            elif el == "or":
                second = self.stack.pop()
                first = self.stack.pop()
                if first in self.var_names:
                    res += "\tmov eax, dword[" + first + "]\n"
                else:
                    res += "\tmov eax, 0x" + single(first).hex()[0] +
"\n"
                if second in self.var_names:
                    res += "\tor eax, dword[" + second + "]\n"
```

```
                else:
                    res += "\tor eax, 0x" + single(second).hex()[0] +
"\n"
                res += "\tmov dword[RES], eax\n"
                self.calced = True
            elif el == "output":
                out_vars = []
                count = 0
                prev = self.stack.pop()
                while prev == ",":
                    count += 1
                    prev = self.stack.pop()
                out_vars.append(prev)
                for _ in range(count):
                    out_vars.append(self.stack.pop())
                for var in out_vars[::-1]:
                    res += "\tPRINT_HEX 4, "+var+"\n\tNEWLINE\n"
            elif el == "read":
                out_vars = []
                count = 0
                prev = self.stack.pop()
                while prev == ",":
                    count += 1
                    prev = self.stack.pop()
                out_vars.append(prev)
                for _ in range(count):
                    out_vars.append(self.stack.pop())
                for var in out_vars[::-1]:
                    res += "\tGET_HEX 4, "+var+"\n"
            elif el[0] == "[" and (self.main_rpn[i+1] == "if" or
self.main_rpn[i+1] == "elseif"):
                res += "\tfinit\n"
                res += "\tfld dword[RES]\n"
                res += "\tmov dword[TMP], 0x" +
single(str(0)).hex()[0] + "\n\tfld dword[TMP]\n"
                res += "\tfcomi\n"
                res += "\tjnz label"+str(self.labels_count)+"\n"
                self.labels_count += 1
```

```
                res += "\tjmp "+el[1:-1:]+"\n"
                res += "label"+str(self.labels_count-1) + ":\n"
                self.calced = False
            elif el[0] == "[" and self.main_rpn[i+1] == "for":
                second = self.stack.pop()
                first = self.stack.pop()
                res += "\tfinit\n"
                res += "\tfld dword["+first+"]\n"
                if second in self.var_names:
                    res += "\tfld dword["+second+"]\n"
                else:
                    res += "\tmov dword[TMP], 0x" +
single(second).hex()[0] + "\n\tfld dword[TMP]\n"
                res += "\tfcomi\n"
                res += "\tjz " + el[1:-1:] + "\n"
            elif el[0] == "[" and self.main_rpn[i+1] == "while":
                res += "\tfinit\n"
                res += "\tfld dword[RES]\n"
                res += "\tmov dword[TMP], 0x" +
single(str(0)).hex()[0] + "\n\tfld dword[TMP]\n"
                res += "\tfcomi\n"
                res += "\tjz "+el[1:-1:]+"\n"
            elif el[0] == "[":
                res += "\tjmp "+el[1:-1:]+"\n"
            elif el[0] == "(":
                res += el[1:-1:] + ":\n"
            else:
                self.stack.append(el)
        res += "\tret\n"
        return res
```

 # gui.py

```
import PySimpleGUI as sg
import io
import os
import lexer
import matrix
import parser
import semantic
```

```
import RPN
import ASM
layout = [[sg.Text("Choose a file: "), sg.Input(),
sg.FileBrowse(key="-IN-FILE-"), sg.Button("Load"),
sg.Button('Compile', size=(12, 2), key="_COMPILE_"), sg.Button('Build
EXE', size=(12, 2), key="_BUILD_")],
          [sg.Text("Source code:", size=(46, 1)), sg.Text("Lexemes:",
size=(45, 1)), sg.Text("Reverse Polish notation:", size=(45, 1)),
sg.Text("Assembler (NASM):", size=(45, 1)),],
          [sg.Multiline(s=(50, 30), key="-INPUT-"),
sg.Multiline(s=(50, 30), disabled=True, key="-LEXER-"),
sg.Multiline(s=(50, 30), disabled=True, key="-RPN-"),
sg.Multiline(s=(50, 30), disabled=True, key="-ASM-")],
          [sg.Text("Errors:")],
          [sg.Multiline(s=(50, 7), key="-ERROR-", disabled=True)]]
window = sg.Window('Compiler', layout)
def do_compile(stream):
    lex = lexer.Lexer(stream)
    operators = []
    operators_semantic = []
    operator_values = []
    res = ""
    err = ""
    while lex.symbol != lexer.Lexer.EOF:
        lex.next_token()
        if lex.error:
            break
        operators.append(lexer.decrypt_to_operators[lex.symbol])
        if lex.symbol == lexer.Lexer.ID or lex.symbol ==
lexer.Lexer.NUM or lex.symbol == lexer.Lexer.REAL or lex.symbol ==
lexer.Lexer.BOOL_VAL:
            operators_semantic.append(semantic.Operator(lex.symbol,
lexer.decrypt_to_operators[lex.symbol], lex.value))
            operator_values.append([lexer.decrypt[lex.symbol],
lex.value])
        else:
            operators_semantic.append(semantic.Operator(lex.symbol,
lexer.decrypt_to_operators[lex.symbol], None))
```

```
            operator_values.append([lexer.decrypt[lex.symbol]])

        if lex.symbol == lexer.Lexer.ID:
            res += lexer.decrypt_to_lexemes[lex.symbol] + ": " +
str(lex.value) + "\n"
        elif lex.symbol == lexer.Lexer.ID or lex.symbol ==
lexer.Lexer.NUM or lex.symbol == lexer.Lexer.REAL:
            res += "CONSTANT: " + lexer.decrypt_to_lexemes[lex.symbol]
+ "," + str(lex.value) + "\n"
        elif lex.symbol in lexer.Lexer.SYMBOLS.values() or lex.symbol
== lexer.Lexer.EOF:
            res += "SYMBOL: " + lexer.decrypt_to_lexemes[lex.symbol] +
"\n"
        else:
            res += "WORD: " + lexer.decrypt_to_lexemes[lex.symbol] +
"\n"
    if lex.error:
        window["-ERROR-"].update(lex.error_msg)
        return
    window["-LEXER-"].update(res)
    err += lex.error_msg + "\n"
    stream.close()
    source_path = "grammar.txt"
    stream = open(source_path, 'r')
    gen = matrix.MatrixGenerator(stream)
    gen.generate()
    if gen.error:
        window["-ERROR-"].update(gen.error_msg)
        return
    err += gen.error_msg + "\n"
    stream.close()
    rules = gen.rules
    for rule_name in rules.keys():
        for rule in rules[rule_name]:
            for i in range(len(rule)):
                if rule[i] not in gen.T:
                    rule[i] = "E"
```

```
    pars = parser.Parser(gen.operator_matrix, operators, rules)
    pars.check()
    if pars.error:
        window["-ERROR-"].update(pars.error_msg)
        return
    err += pars.error_msg + "\n"
    sem = semantic.Semantic(operators_semantic)
    sem.check()
    if sem.error:
        window["-ERROR-"].update(sem.error_msg)
        return
    err += sem.error_msg + "\n"
    rpn = RPN.RPN(operator_values)
    declare_res, res = rpn.convert()
    if rpn.error:
        window["-ERROR-"].update(rpn.error_msg)
        return
    res_string = ""
    for el in declare_res:
        res_string += str(el) + " "
    for el in res:
        res_string += str(el) + " "
    window["-RPN-"].update(res_string)
    err += rpn.error_msg + "\n"
    asm_gen = ASM.ASM(declare_res, res)
    res_string = asm_gen.generate()
    if asm_gen.error:
        window["-ERROR-"].update(asm_gen.error_msg)
        return
    window["-ASM-"].update(res_string)
    err += asm_gen.error_msg + "\n"
    window["-ERROR-"].update(err)
while True:
    event, values = window.read()
    if event == sg.WINDOW_CLOSED:
        break
    if event == "_COMPILE_":
        do_compile(io.StringIO(values["-INPUT-"]))
```

```
    elif event == "Load":
        if values["-IN-FILE-"] != "":
            file = open(values["-IN-FILE-"])
            window["-INPUT-"].update(file.read())
    elif event == "_BUILD_":
        asm_file = open("asm.asm", "w")
        asm_file.write(values["-ASM-"])
        asm_file.close()
        os.system('.\\NASM\\nasm.exe --gprefix _ -f win32 asm.asm -o
asm.obj')
        os.system(".\MinGW\\bin\gcc.exe asm.obj .\\NASM\macro.o -g -o
asm.exe -m32")
window.close()
```